GECKOP: Gestion Económica de Proyectos de Investigación

Por
Azahara Fernández Notario
Isabel Núñez De La Torre

UNIVERSIDAD COMPLUTENSE
MADRID

Grado en Ingeniería Informática
Facultad de Informática

Mercedes García Merayo
GECKOP: Gestion Económica de Proyectos de Investigación

Madrid, 2018–2019

Agradecimientos

Agradecer a mi familia, pareja y amigos el apoyo incondicional durante toda la etapa universitaria. En especial mencionar a Hen que se ha quedado las noches en vela acompañándome durante el trabajo, a Irene que ha soportado todas mis quejas unas veces mas justificadas que otras y a Isa mi compañera y complemento durante este trabajo

Azahara Fernández

Quiero agradecer a mi familia, hermano y amigos por su apoyo y ánimos día a día, a Miguel por su ayuda y comprensión. En especial agradecer a mi compañera Aza el esfuerzo y las horas invertidas en este arduo proyecto, sin su ayuda no hubiera sido posible finalizar este trabajo de manera tan profesional.

Isabel Núñez

A Manolo por darnos una opinión más. Y a nuestra tutora Mercedes por guiarnos en este camino y ayudarnos a finalizar esta etapa tan dura de la Universidad.

Las autoras.

Índice general

Resumen

Este trabajo consiste en el desarrollo de una aplicación web que facilite la tramitación de las órdenes de pago de los proyectos de investigación. Esta plataforma permite la centralización de la gestión económica de los gastos. Por una parte, facilita a los miembros de los proyectos generar las órdenes de pago correspondientes a sus gastos y enviarlas electrónicamente al investigador principal correspondiente para la aceptación y firma. Por otra parte apoya a los investigadores principales en el seguimiento económico de los proyecto mediante el almacenamiento y centralización de la información correspondiente a los gastos. Para el desarrollo se han utilizado tecnologías como Angular y Spring, que actualmente son de las más punteras en el mercado del desarrollo web, completando así la parte correspondiente a la investigación y formación que comprende el desarrollo de un TFG.

Palabras Clave
Aplicación Web, Orden de Pago, Investigador Principal, Angular, Spring, Proyecto.

Abstract

This work aims to develop a web application to ease the economic activities of the research projects in the informatics faculty. From this platform we seek to actively help in the setup and management for the members in the projects and their roles, in correlation of their participation on said project, namely: Lead Investigator, member of the project or guest teacher. Moreover, we could manage the payment orders, coulding these be accepted, rejected, modified, and so on. We must highlight the simplicity to manage and arrange the creditors whom part of the budget is dedicated. For the development, we applied technologies as Angular and Spring, which are currently ones of the most advanced in the field, thus completing the share appropriate to the research and training to the development of a Final Degree Proyect.

Keywords
Web Application, Payment Order, Lead Investigator, Angular, Spring, Project.

Capítulo 1

Introducción

Este documento pretende recoger todos los aspectos fundamentales para comprender la funcionalidad de la aplicación web creada como Trabajo de Fin de Grado para facilitar tramitación de las órdenes de pago correspondientes a los proyectos de investigación gestionados por la Fundación General de la UCM.
El código de la aplicación está subido a los repositorios de github:

- Para el Front-End: https://github.com/IsaNuTor/geckop-copia
- Para el Back-End https://github.com/IsaNuTor/spring-backend-geckop

1.1. Motivación

La aplicación desarrollada pretende renovar, mejorar y ser implantada en sustitución de la usada en la actualidad para cubrir los fines desarrollados en esta.

1.1.1. Mejoras tecnológicas

Para la mejora a nivel tecnológico se ha decidido utilizar tecnologías punteras en el mercado, destacando entre ellas los principales *frameworks* que permiten dar soporte y estructura a la aplicación:

Angular para el desarrollo del *front-end*, por ser de código abierto y permitir una aplicación mucho más modular y mantenible, además de facilitar el desarrollo con el uso de TypeScript, un lenguaje mucho mas intuitivo y versátil que su predecesor JavaScript.

Spring Boot para el desarrollo del *back-end*, por permitir la integración del patrón Modelo-Vista-Controlador, poder desarrollar en Java y en general adaptar el desarrollo web a tecnologías de uso común para así facilitar futuras mejoras y el mantenimiento de la aplicación.

1.1.2. Mejoras visuales

A nivel estético se ha optado por un aspecto mucho mas limpio para conseguir modernizar y hacer más intuitiva la aplicación.

Se ha utilizado Bootstrap para darle estética y manejabilidad, ya que permite el desarrollo de webs *responsive* que se adaptan a cualquier tipo de dispositivo y confiere un aspecto más limpio e intuitivo tanto a nivel de código como de la propia aplicación. La diferencia puede apreciarse en las Figuras 1.1 y 1.2.

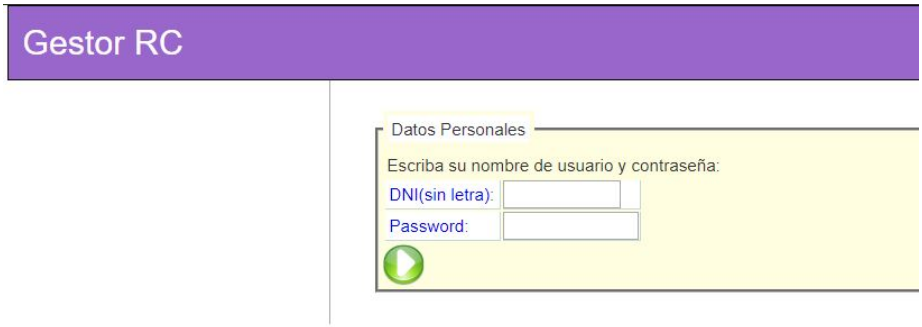


Figura 1.1: Antigua pantalla de login.

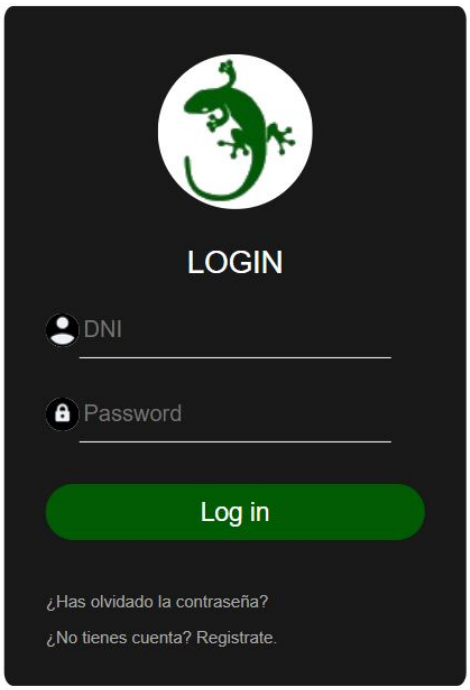


Figura 1.2: Nueva pantalla de login.

1.2. Funcionalidades de la aplicación

Los objetivos principales de la aplicación son:

Órdenes de pago

- Confeccionar órdenes de pago.
- Gestionar las órdenes de pago (rechazarlas, aceptarlas o pedir modificación).
- Emitir el archivo pdf de las órdenes de pago aceptadas.

Proyecto

- Crear un proyecto y administrar sus participantes.
- Editar cada uno de los proyectos de manera correspondiente.

Acreedores

- Añadir los distintos acreedores que corresponden a las órdenes de pago.
- Modificar los datos relevantes de cada acreedor.
- Eliminar aquellos acreedores que no estén asociados a ninguna orden.

Usuarios

- Crear usuarios de la aplicación.
- Gestionar el perfil de cada uno de los usuarios.

Capítulo 2

Análisis de requisitos

2.1. Requisitos

2.1.1. Registro

- Registrar información de un usuario
- El campo DNI tiene que tener ocho números y una letra.
- El campo e-mail tiene que tener formato de correo electrónico válido.
- El usuario una vez se haya registrado, será redirigido al login.
- El usuario no se puede registrar con el mismo DNI de otro usuario ya registrado en la aplicación.
- Para que el registro sea válido, ningún campo del formulario de registro puede estar vacío.
- La contraseña requiere un mínimo de cinco caracteres y un máximo de diez caracteres.
- Para registrarse con éxito el usuario tiene que aceptar obligatoriamente los términos, condiciones y políticas de privacidad.

2.1.2. Login

- Solamente podrán hacer login con éxito los usuarios registrados.
- Si al hacer login la contraseña no es correcta, se notificará al usuario con un error de contraseña incorrecta.
- Si alguien intenta acceder desde las rutas sin estar logueado en la aplicación, redirigirá al login.

2.1.3. Perfil

- El usuario tiene que estar logueado para modificar su perfil.
- El campo IBAN tiene máximo treinta y cuatro caracteres.

2.1.4. Acreedores

- El usuario tiene que estar logueado para crear/editar acreedores.
- No se pueden crear Acreedores con el mismo NIF.
- No se puede editar el campo NIF del acreedor.
- Se pueden eliminar acreedores.

2.1.5. Proyectos

- Para acceder a los proyectos, el usuario tiene que estar logueado en la aplicación.
- Solamente se pueden borrar aquellos proyectos que no tengan ninguna orden asociada.
- La fecha de fin tiene que ser posterior a la fecha de inicio del proyecto.
- La persona que crea el proyecto será el investigador principal de dicho proyecto.
- El número de contabilidad no es requerido.
- El acrónimo no puede ser el mismo en varios proyectos.
- Para asignar un posible segundo investigador principal, tiene que aparecer previamente entre los investigadores del proyecto.
- El campo presupuesto es numérico únicamente.
- Los proyectos únicamente los pueden editar los investigadores principales.

2.1.6. Órdenes

- En la vista de órdenes se mostrarán las órdenes del usuario que está logueado.
- La modificación de la orden solo la puede solicitar el investigador principal del proyecto sobre el que se haya generado dicha orden.
- El usuario solo puede modificar la orden si el investigador principal pide una modificación de esa orden.
- Los datos del investigador principal se añaden cuando se firma la orden.
- El fichero correspondiente (pdf) puede verse únicamente cuando la orden está aceptada.
- Las órdenes de pago tienen asignado un número único en cada proyecto.
- El número máximo de gastos que pueden incluirse en una orden de gastos generales es cinco.
- El número mínimo de gastos que pueden incluirse en una orden de gastos generales es uno.

- El importe total de una orden de viajes tiene que ser mayor que cero.
- Los campos de importes de los gastos son de tipo numérico.
- El tamaño máximo de las imágenes de las facturas es 10MB.
- No se pueden eliminar órdenes de pago.

Capítulo 3

Entornos de Desarrollo

3.1. Frameworks y Librerías

Para organizar la estructura del proyecto hemos utilizado un patrón modelo vista controlador (MVC) representado en la figura 3.1, utilizando varios *frameworks* preparados para su implementación.

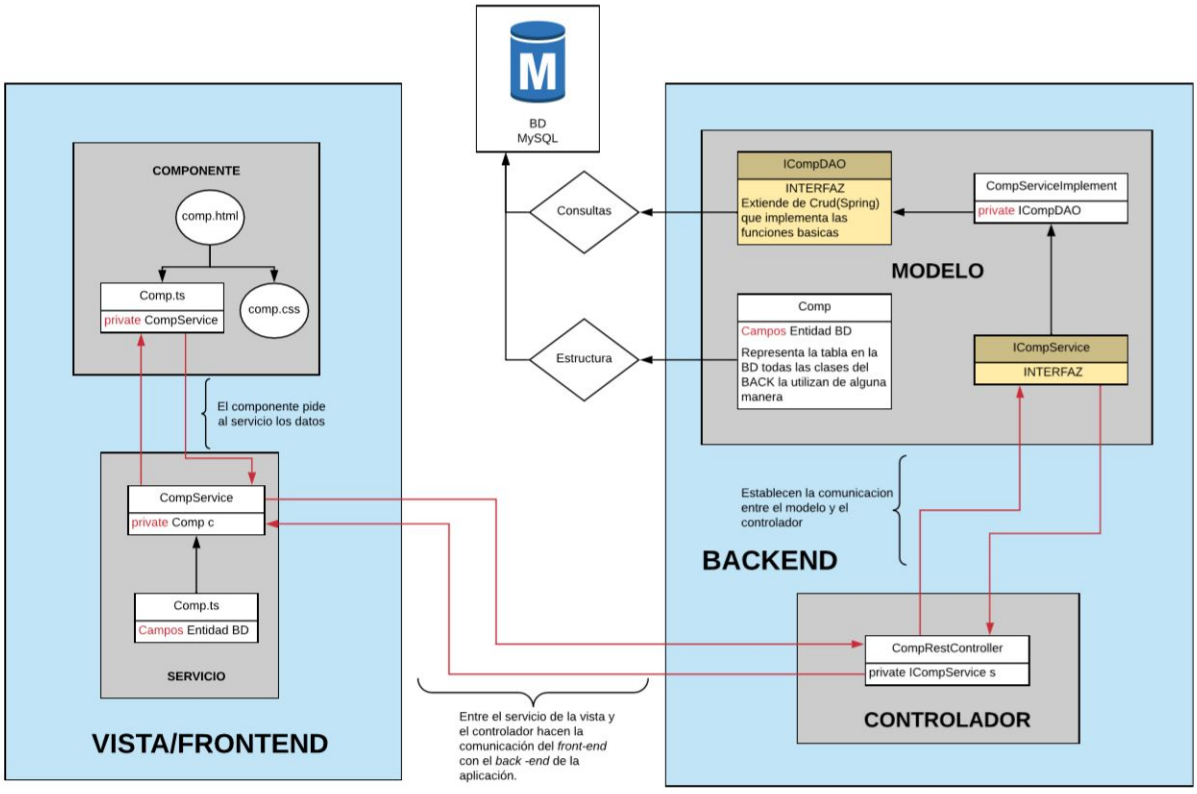


Figura 3.1: Representación del patrón MVC.

3.1.1. Angular

Angular[3] es un *framework* de desarrollo *front-end* de código abierto, desarrollado por Google, que a través del lenguaje Typescript permite la creación de aplicaciones SPA (*Single Page App* o aplicaciones de una sola página) mediante el uso de AJAX [1]. Esto mejora notablemente la velocidad de carga y el manejo de la aplicación evitando continuas recargas del navegador. Además el uso de componentes ayuda a comprender la estructura del código lo que facilita su mantenimiento y legibilidad.

3.1.2. Bootstrap

Bootstrap[7] es un *framework* CSS desarrollado por Twitter y actualmente en mantenimiento y recibiendo mejoras a través de GitHub, que utilizando las propias funcionalidades de HTML5 y CSS permite dar estilo a la aplicación. Utiliza un sistema de 12 columnas, permitiendo así un diseño *responsive*, el cual se adapta a cualquier tipo de pantalla o dispositivo.

3.1.3. Spring Boot

Spring Boot[6] es un *framework* especializado en el desarrollo de aplicaciones web que permite simplificar la puesta en producción de la misma y permite el uso del patrón MVC. En este caso implementado desde Java. Spring se encarga de controlar las posibles dependencias en el desarrollo y las actualiza automáticamente además de implementar un servidor propio. Las peticiones realizadas a través de este servidor son de tipo REST facilitando el manejo de estados del protocolo http y enviando los datos solicitados por cada una de las peticiones en un archivo JSON.

3.1.4. Apache PDFBox

Apache PDFBox[2]es una biblioteca de Apache desarrollada para Java con la que se pueden manejar ficheros PDF y en nuestro caso rellenar los formularios que impone la Fundación UCM para las órdenes de pago.

3.2. Lenguajes de Programación

Para la implementación se han utilizado diferentes tecnologías y lenguajes que se recogen a continuación.

3.2.1. CSS (*Cascading Style Sheets*)

CSS es un lenguaje que permite el desarrollo del estilo de una pagina web cuando es utilizado sobre otro lenguaje de marcado como en este caso HTML. Aunque la mayor parte del aspecto visual de la aplicación está realizado a través de Bootstrap se han incluido algunas hojas de estilo para añadir detalles que este *framework* no contempla.

[1] Tecnología que permite el desarrollo de aplicaciones interactivas

3.2.2. HTML

HTML es un lenguaje de marcado, que permite de visualizar paginas web. Es el estándar mundial para este propósito.

3.2.3. Java

Java es uno de los más conocidos lenguajes de programación orientado a objetos. Las aplicaciones construidas en Java pueden ejecutarse en cualquier máquina de Java (JVM).

3.2.4. TypeScript

TypeScript es un lenguaje de programación, desarrollado por Microsoft y derivado de JavaScript. Actualmente es el que utiliza Angular para el desarrollo de la funcionalidad de sus componentes. El uso de TypeScript requiere de jQuery para su buen funcionamiento. Está es una de las principales bibliotecas de JavaScript que permite modificar las secciones de HTML y utilizar AJAX de manera más cómoda.

3.3. Base de Datos

Puesto que la aplicación requiere de una sola base de datos se ha escogido MySQL como Sistema Gestor de Base de Datos (SGBD), el cual nos permite gestionar los datos que recoge la aplicación y almacenarlos.

3.4. Control de Versiones

Para mantener un correcto control de versiones y un entorno de trabajo profesional se ha utilizado la herramienta GitHub.
GitHub[1] es una herramienta de control de versiones que se apoya en la tecnología de Git para mantener un repositorio ordenado y que controla todos los cambios que se van haciendo en el proyecto. Además, mantiene una copia de seguridad en la nube de todo aquel trabajo realizado en local para evitar posibles pérdidas. En las figuras 3.2 y 3.3 se puede ver la pantalla de los dos proyectos necesarios para el desarrollo desde la versión web de GitHub.

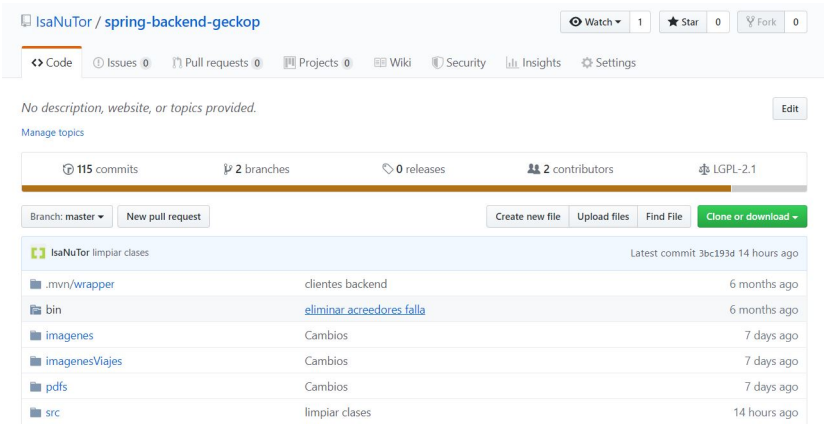


Figura 3.2: Vista Github proyecto back-end.

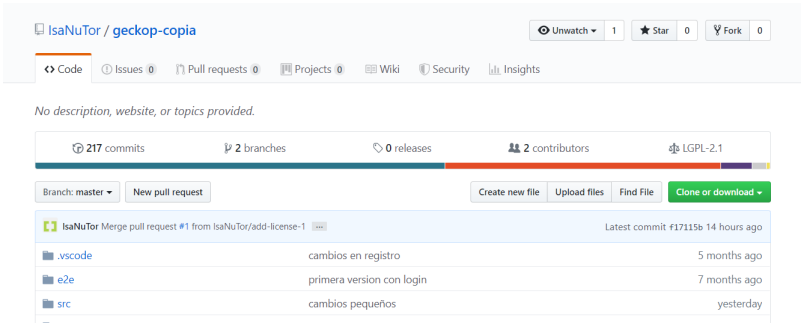


Figura 3.3: Vista Github proyecto front-end.

3.5. Maquetación y Prototipos

Para el desarrollo de maquetas y prototipos que ayudaron a obtener una visión general de como debería quedar el aspecto visual y la navegabilidad por la aplicación. Adobe XD es una herramienta de Adobe cuya versión gratuita para principiantes fue suficiente para este pretexto.

Capítulo 4

Diseño e Implementación

El desarrollo de este proyecto se ha ido realizando en distintas fases durante el segundo cuatrimestre y los meses de verano del curso 18-19.

4.1. Fase de análisis

Cuando se planteó el proyecto los principales problemas a resolver eran: actualizar la estética de la aplicación, eliminar funcionalidades obsoletas y añadir aquellas que habían ido surgiendo a lo largo del uso de la aplicación, así como implementar un modulo capaz de rellenar los formularios PDF necesarios para tramitar las órdenes de pago.
Tras el análisis del estado en el que se encontraba la aplicación se decidió diseñar una nueva con tecnologías actuales que permitiesen resolver los problemas y en un futuro facilitar el mantenimiento que no se había dado hasta el momento.

4.2. Fase de investigación

Durante la fase de análisis se seleccionaron las distintas tecnologías que permitiesen resolver los problemas presentados en el análisis. Tras estudiar el posible uso de aquellas ya aprendidas a lo largo de la formación universitaria se decidió que si se quería que la aplicación fuese mantenible había que utilizar herramientas más actuales y con futuro en el mercado. Por ello, al no haber estudiado antes dichas tecnologías, esta ha sido una de las fases más importantes y costosas. En ella se han realizado un arduo proceso de formación personal y pequeños proyectos de prueba con cada una de las tecnologías hasta conseguir un dominio fluido de las mismas para poder dedicarse al proyecto real.
Uno de los mayores problemas encontrados ha sido la falta de bibliotecas y material para trabajar con archivos de tipo PDF. Durante la fase de búsqueda solo se habían encontrado bibliotecas de pago. A proceder a la instalación de la versión de prueba de una de ellas (PDFFields del desarrollador Qoppa) buscando como añadirla a nuestro directorio de dependencias de Maven encontramos una gratuita de Apache que finalmente es la que se ha utilizado.

4.3. Fase de Instalación y Configuración

En la fase de instalación y configuración se preparó todo el entorno de desarrollo. Para ello se han realizado los siguientes pasos.

Paso 1 - Utilización de Git y creación de un proyecto en GitHub: Para mantener el control de versiones y copias de seguridad constantes del proyecto se utiliza Git y junto a Github se facilita su uso.

Paso 2 - Instalación del IDE[1] Eclipse: Descarga del programa de código abierto desde la propia web del proveedor y la extensión adecuada para el desarrollo de Spring Boot.

Paso 3 - Instalación de Node.js y Angular: Descarga e instalación de Node.js[2], este incluye el componente npm (Node package manager) desde donde se descargan y administran las dependencias, es decir, los módulos y librerías de Angular. Después se descarga Angular para crear la aplicación web, las clases, los componentes y los servicios que contiene la aplicación creada.

Paso 4 - Creación del proyecto de Spring: En Eclipse para poder lanzar el servidor y ejecutar las consultas hacia la base de datos y realizar todo su desarrollo *back-end*. Primero una vez instalado eclipse e instalamos Spring Tools 4. A continuación procedemos a crear un proyecto Spring Starter Project. El tipo de proyecto en nuestro caso será Maven, con la posibilidad de despligue en un servidor externo. Seleccionamos las dependencias, es decir librerías que incluimos en nuestro proyecto, las cuales podremos ver en el archivo pom.xml una vez creado el proyecto.

Paso 5 - Descarga y configuración de MySQL: Descarga MySQL Workbench[3] desde la propia web del proveedor con licencia GPL, se seleccionan los productos a instalar MySQL Server 8.0 y MySQL Workbench, dejamos por defecto la configuración Standalone MySQL Server/Classic MySQL Replication, username es root por defecto y le introducimos una contraseña, como nombre de servicio se pone MySQL80. Para el funcionamiento correcto en local hay que configurar en spring el proyecto. En el archivo application.properties hay que configurar la ruta, el nombre de usuario y la contraseña de la base de datos para que al lanzar el proyecto sepa donde tiene que cargar la base de datos.

Paso 6 - Creación del proyecto en Angular: Para poder ejecutar la aplicación en navegador y realizar todo su desarrollo. Con el comando ng new geckop creamos nuestro proyecto *front-end*

Paso 7 - Introducción e instalación de bibliotecas y módulos en Angular: Inclusión de algunas bibliotecas necesarias o módulos útiles de Angular c. Las librerías y módulos incluidos son:

- Bootstrap
- jQuery

[1]IDE-Entorno de desarrollo integrado.
[2]Más referencias en: nodejs.org
[3]Más referencias en:dev.mysql.com/downloads/workbench

- Cloudflare: requerida por Bootstrap.
- Auth0: requerida por Bootstrap.
- SweetAlert2[8]: Para mejorar el estilo de las alertas que se emiten al usuario, reflejado en la Figura 4.1.
- Rxjs[4]: Para implementar los Observables y facilitar la lectura y redacción de código asíncrono.
- Forms Validator (@angular/forms): Para implementar las validaciones correspondientes de cada uno de los formularios.
- Router (@angular/router): Para poder moverse entre rutas, porque aunque la web pretende explotar el uso de AJAX y SPA se ha utilizado una pagina por cada vista y estas explotan individualmente esta tecnología.

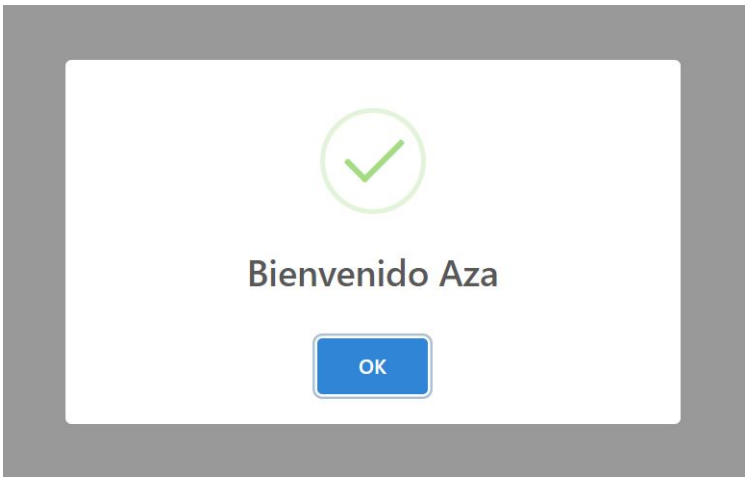


Figura 4.1: Ejemplo de alerta con SweetAlert2.

Una vez descargados, instalados y configurados todos los programas librerías y herramientas necesarias se pudo proceder al desarrollo de la aplicación, aunque algunas de las librerías y módulos del paso 7 se han ido añadiendo a lo largo del desarrollo.

4.4. Fase de Diseño

Durante el diseño se llevaron a cabo distintas maquetas del aspecto visual con Adobe XD, se puede ver un ejemplo en la Figura 4.2. Para ello se analizaron las distintas vistas que posee la aplicación. Una vez realizada esta fase previa de diseño, se realizaron las vistas definitivas que se muestran en los apartados posteriores.

[4]Más referencias en: rxjs-dev.firebaseapp.com

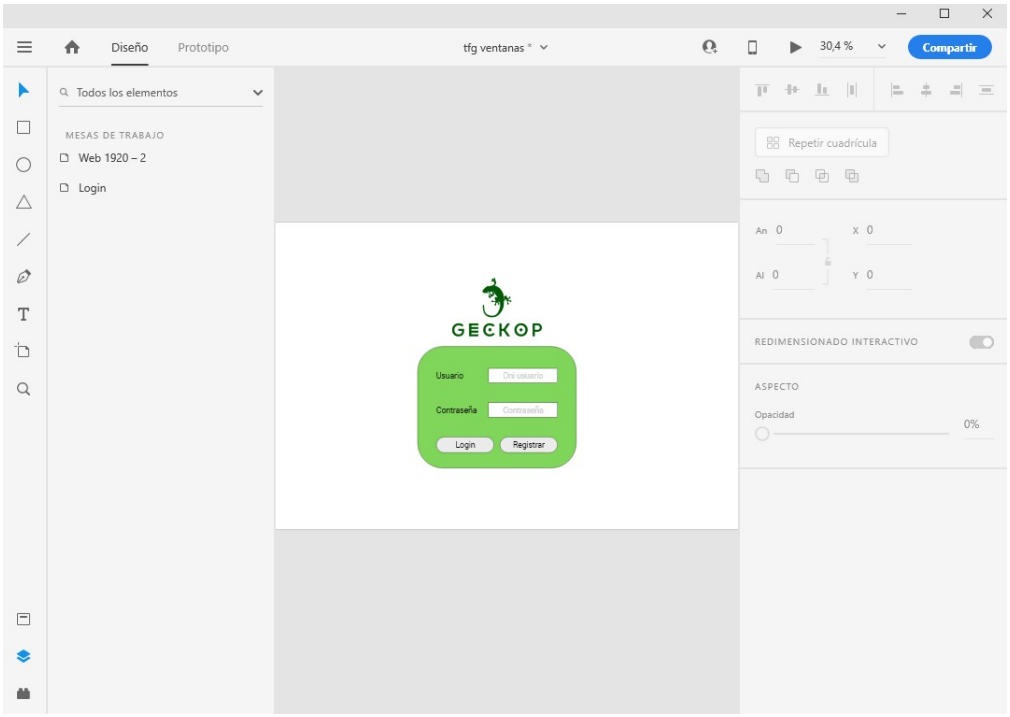


Figura 4.2: Mockup Login desde Adobe XD.

4.4.1. Sistema de Usuarios

El sistema de usuarios de la aplicación es aquel que se encarga de gestionar, crear y/o modificar los datos de cada usuario. Se compone de tres pantallas principales: vista de registro, login y perfil.

Vista de Registro

En la vista del registro se presenta un formulario donde se solicita el DNI/NIF, nombre y apellidos, correo electrónico, teléfono, departamento, centro y contraseña del usuario, que se procede a almacenar en la base de datos registrando así al usuario en la aplicación. Tras el registro se redirige a la pantalla de login, donde puede proceder a iniciar sesión. Figura 4.3.

Figura 4.3: Vista del registro.

Vista de Login

En la vista de login se presenta un formulario donde iniciar sesión en el que solicita el DNI/NIF del usuario y su contraseña. Tras el inicio de sesión se redirige a la pantalla principal de la aplicación. Figura 4.4.

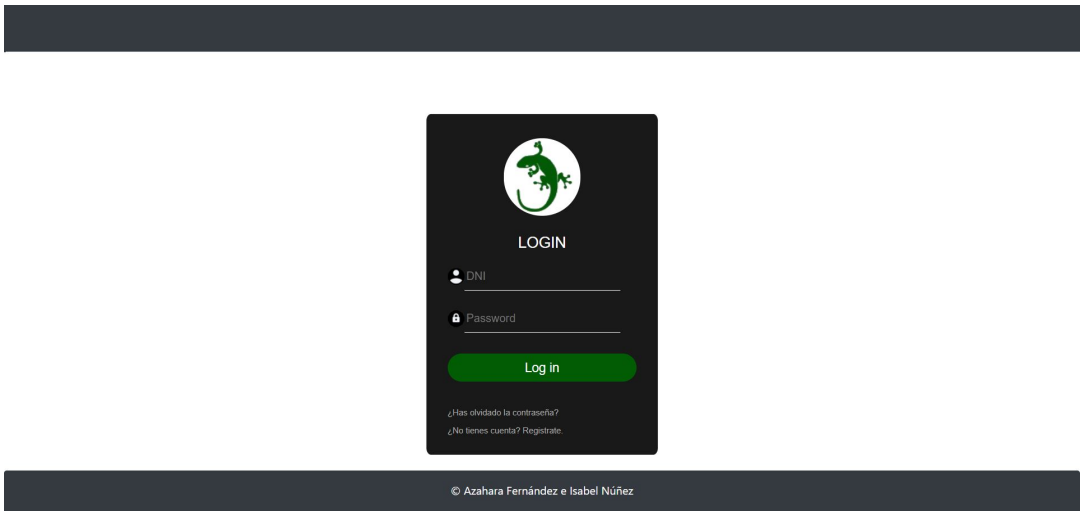


Figura 4.4: Vista del login.

Vista Principal

Cuando se accede a la aplicación aparece una pantalla del estado actual de las órdenes del usuario. Se compone de dos pestañas. También se puede acceder a la vista pulsando el icono Geckop de la barra de navegación desde cualquiera de las vistas una vez logueado.

- **Pendientes de firma:** Muestra las órdenes de aquellos proyectos en los que el usuario participa como investigador principal (IP). La Figura 4.5 muestra la vista de Home.

- **Mis órdenes:**

- **Aceptadas:** Muestra aquellas órdenes emitidas por el usuario que ya han sido revisadas y aceptadas por el IP del proyecto correspondiente.
- **Rechazadas:** Muestra aquellas órdenes emitidas por el usuario que ya han sido revisadas y rechazadas por el IP del proyecto correspondiente.
- **Pendientes:** Muestra las órdenes que ha emitido el usuario y están pendientes de firmar por el IP del proyecto al que pertenezca dicha orden.
- **Pendientes de revisión:** Muestra las órdenes que ha emitido el usuario y requieren alguna modificación que ha señalado el IP del proyecto al revisar la orden antes de firmar.

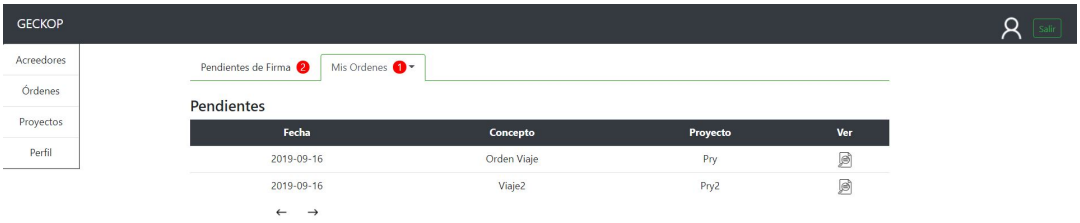


Figura 4.5: Vista Home.

Vista de Perfil

En la vista de perfil se presentan los datos del usuario y entradas para modificar aquellos que el usuario considere necesarios. Para acceder a esa vista se ha de pulsar el botón “Perfil”del menú lateral. Figura 4.6.

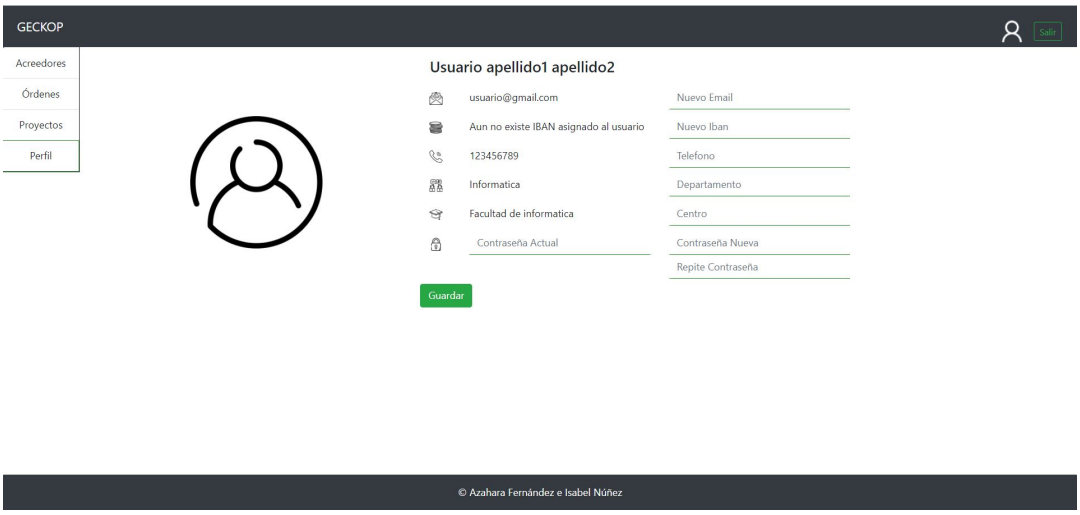


Figura 4.6: Vista del perfil.

4.4.2. Sistema de Acreedores

El sistema de acreedores es el encargado de administrar aquellas personas y entidades que recibirán parte del presupuesto de los proyectos. Por norma general los investigadores y aquellas entidades proveedoras de material.

Vista de Acreedores

En la vista de acreedores se muestra una tabla resumen de los acreedores registrados en la aplicación. Además contiene un botón "Nuevo acreedor" que permite registrar nuevos acreedores. Para acceder a esa vista se ha de pulsar el botón "Acreedores" del menú lateral. Figura 4.7.

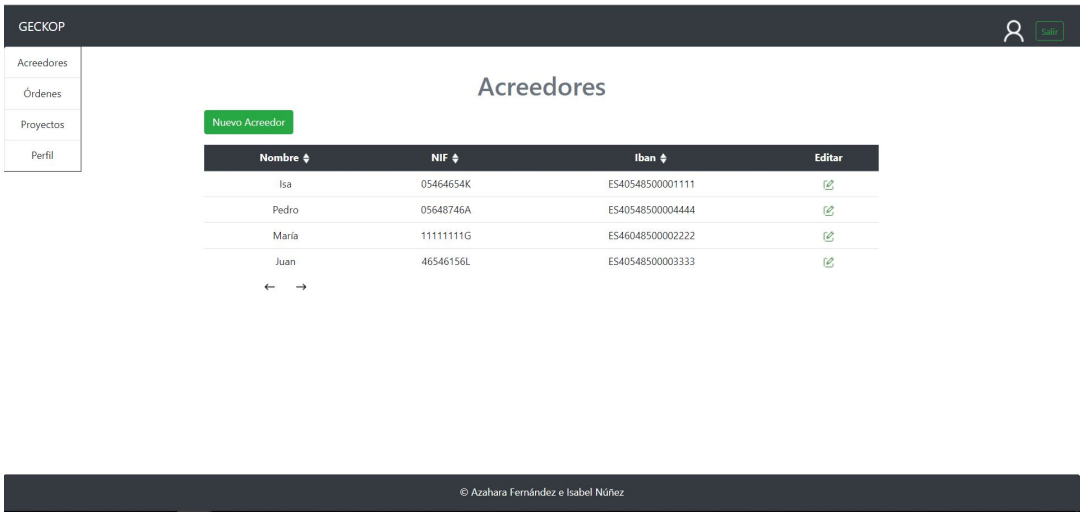


Figura 4.7: Vista de acreedores.

Formulario de Acreedor

Esta vista contiene un formulario a través del cual los propios usuarios de la aplicación pueden registrar nuevos acreedores o modificarlos. Para crear un nuevo acreedor accederemos a esta vista desde el botón "Nuevo acreedor" que permite registrar nuevos acreedores. Para editar un acreedor ya existente accederemos desde el botón "Editar" de cada una de las entradas de acreedor de la tabla. Figura 4.8.

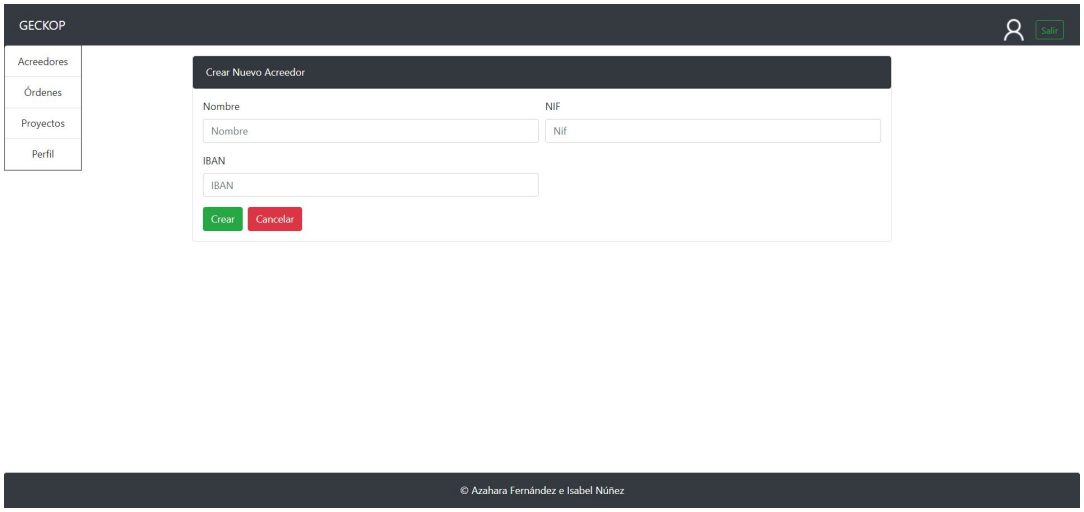


Figura 4.8: Vista del formulario de acreedor.

4.4.3. Sistema de Proyectos

Este sistema es el encargado de crear, ver y editar los proyectos y sus participantes. Los IP serán los encargados de registrar los proyectos y añadir a los correspondientes investigadores.

Vista de Proyectos

La vista de proyectos muestra una tabla en la que aparecen los datos principales de cada proyecto en el que el usuario logueado participa (ya sea como investigador o como IP) con un enlace que permite visualizar o editar cada uno. Para acceder a esa vista se ha de pulsar el botón “Proyectos” del menú lateral. Figura 4.9.

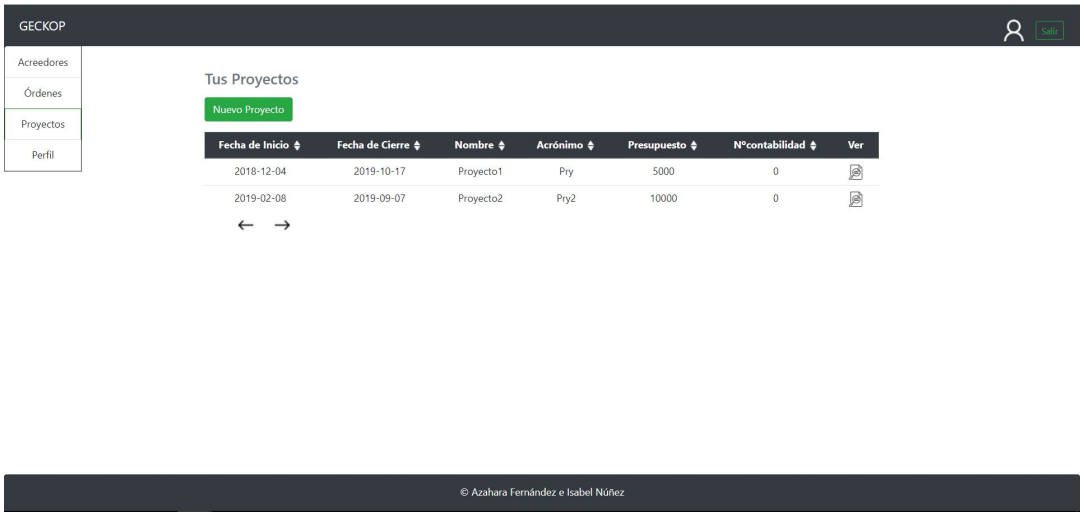


Figura 4.9: Vista de la tabla de proyectos.

Vista Individual de Proyecto

Esta vista permite ver las características de cada proyecto más a fondo y editar algunos de sus datos en caso de ser uno de los investigadores principales de este. Para acceder a

esa vista se ha de pulsar el botón "Ver/Editar" de cada una de las entradas de la tabla principal. Figura 4.10.

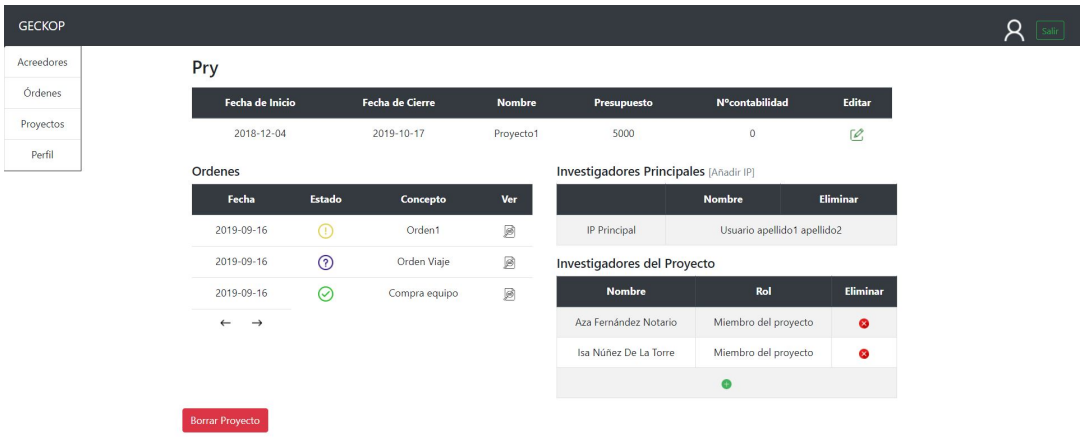


Figura 4.10: Vista individual de cada proyecto.

Formulario de Proyecto

El formulario de cada proyecto contiene las distintas entradas de datos necesarias para crear un proyecto además dos de tablas. En la tabla izquierda se pueden seleccionar los investigadores que formaran parte del proyecto. En la tabla derecha aparecerán aquellos investigadores que ya se hayan añadido y una opción para seleccionar el rol de cada uno, además de un botón que permite eliminarlos. Para acceder pulsamos el botón "Nuevo Proyecto" de la vista de proyectos. Figura 4.11.

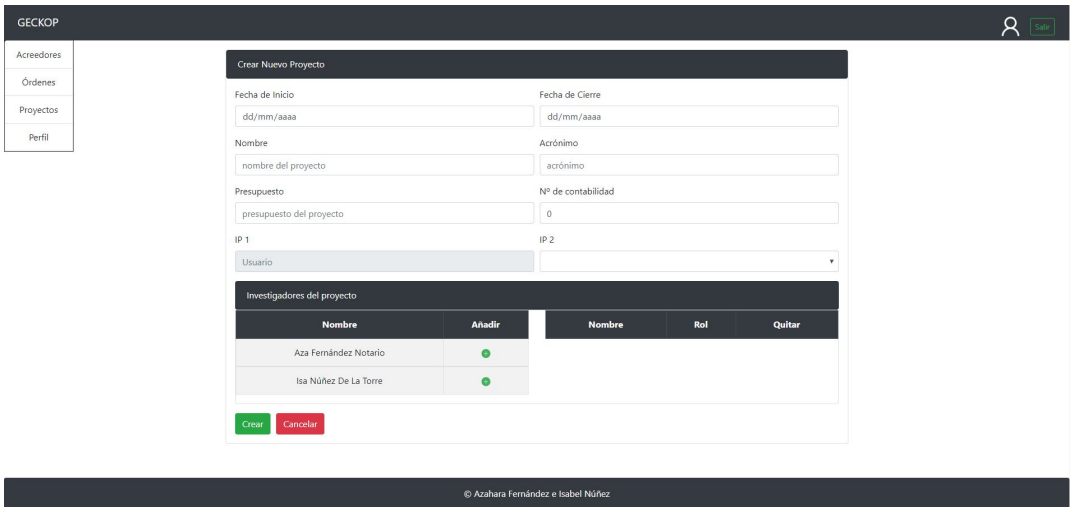


Figura 4.11: Vista final del formulario de creación de proyecto.

4.4.4. Sistema de Órdenes

El sistema de órdenes es el módulo principal de la aplicación. Se encarga de crear, enviar las órdenes de pago, permitir su modificación y firma y visualizarlas cuando se necesita.

Vista Órdenes

En la vista de órdenes se muestra una tabla con las órdenes emitidas por el usuario logueado. Además, contiene un botón "Nueva Orden" que ofrecerá la opción entre los dos tipos de orden: orden de gastos generales u orden de viajes y desplazamientos. Para acceder a esta vista se ha de pulsar el botón "Órdenes" del menú lateral. Figuras 4.12 y 4.13.

Figura 4.12: Vista de órdenes.

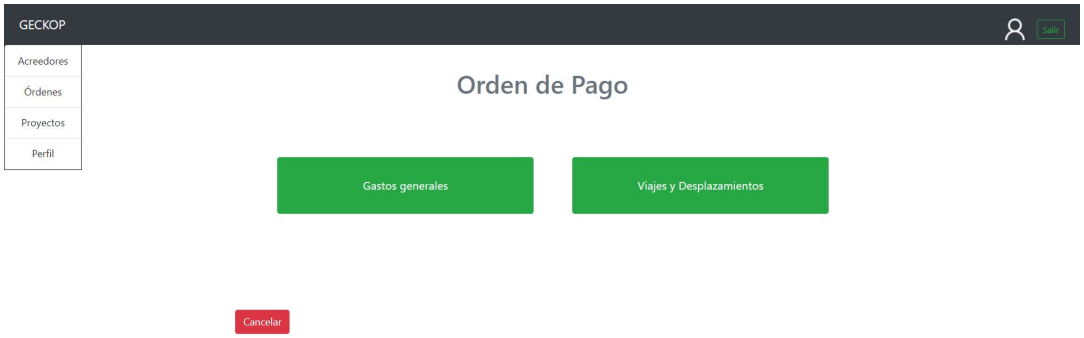


Figura 4.13: Vista nueva orden.

Vista Individual de Orden

Cuando accedemos a una orden de manera individual se ofrecen distintas opciones según el estado de esta. Los estados posibles son: aceptada(A), pendiente(P) o rechazada(R). Si el estado es A o R solamente se puede visualizar, si el estado es P se permite modificar y firmar si el usuario esta autorizado para ello. Hay diversas formas de acceder a esta pantalla. En general desde cualquiera de las demás vistas que contengan un formato resumido de alguna orden. Se puede ver en la Figura 4.14 una orden aceptada y en la Figura 4.15. una orden pendiente de firma.

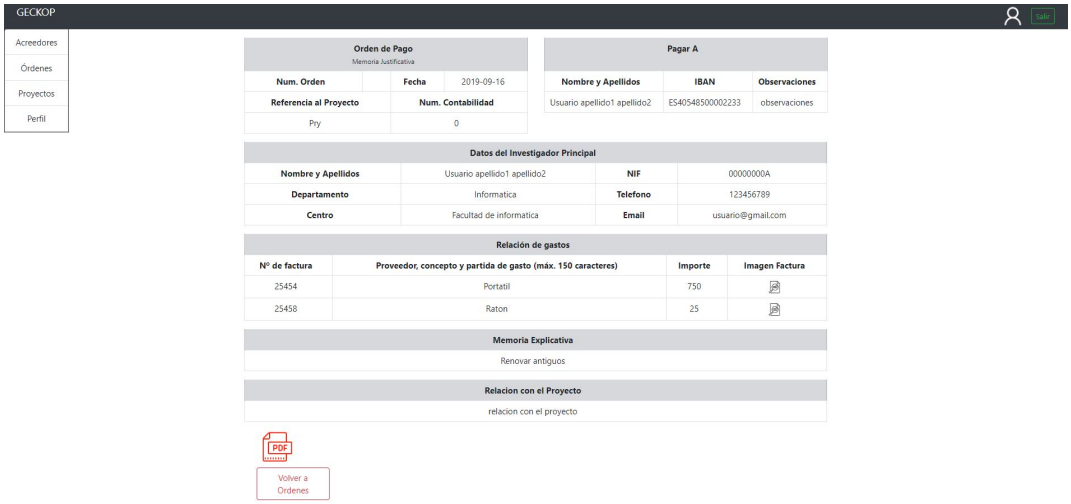


Figura 4.14: Vista de orden aceptada.

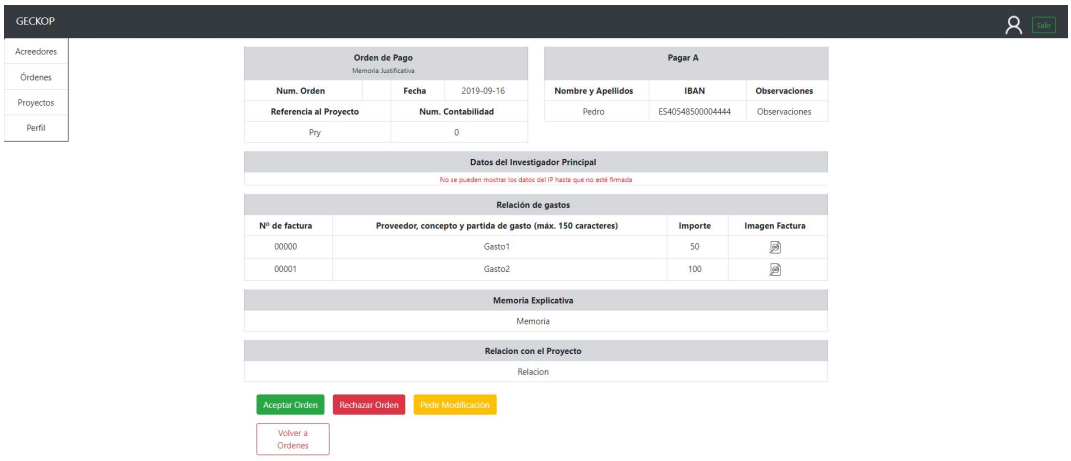


Figura 4.15: Vista de orden pendiente.

Formularios de Orden

Al emitir una orden se presentan dos opciones: orden de gastos generales y orden de viajes y desplazamientos. Cada una de ellas tiene su propio formulario con los campos correspondientes en cada caso. Para acceder a crear una orden pulsamos el botón “Nueva Orden” desde la vista de órdenes. Se pueden ver los dos tipos de orden en las Figuras 4.16 y 4.17

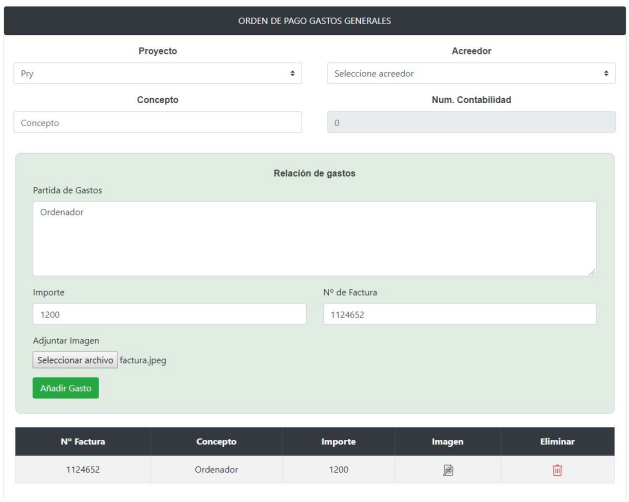


Figura 4.16: Formulario orden de gastos generales.

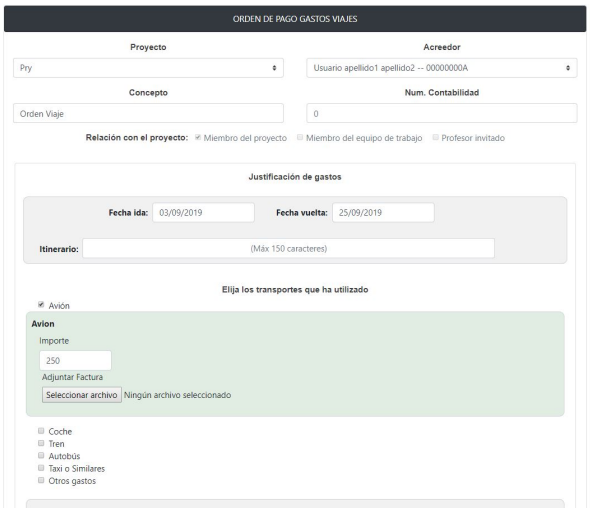


Figura 4.17: Formulario orden de viajes.

4.5. Fase de Desarrollo

En la fase de desarrollo se ha implementado toda la funcionalidad de la aplicación. Este desarrollo se puede dividir en tres etapas: esquematización, implementación y pruebas.

4.5.1. Esquematización

Durante esta etapa se realizó una versión estática en HTML de lo que debía visulizarse tras la implementación de la parte dinámica. Para ello, se tradujo la maqueta de cada una de las vistas a su equivalente en lenguaje HTML. Aunque cada vista trate un tipo de dato concreto a nivel HTML se pueden agregar puesto que son muy similares.

Vistas de Proyecto, Acreedores y Órdenes

Estas vistas están enfocadas a presentar un resumen de cada uno de los elementos principales de la aplicación. Tras valorar distintas maneras de presentar la información

se decidió usar una tabla que contuviese la información más importante de cada objeto. Para ello se utiliza la etiqueta *<table>* de HTML. Otra de las características comunes de estas vistas es un botón de acceso a la creación de un nuevo objeto. A nivel visual se ha utilizado la clase *table* de Bootstrap con la clase *thead-dark* en los encabezados para darle un aspecto más elegante a las tablas de HTML.

Formularios

Para cada uno de los formularios se analizado que objeto era más apropiado para recoger cada uno de los campos. Los principales son *<input>* para recogida de texto simple (máx. 3 palabras), *<select>* para opciones fijas escogidas a través de una lista desplegable y *<textarea>* para textos largos como comentarios o descripciones (máx. 150 caracteres). Para el estilo de los formularios se han utilizado las clases de Bootstrap: *form-group* para los formularios de manera general y *form-control* para cada uno de los elementos de estos.

Vistas individuales

En el caso de las vistas individuales de cada objeto se ha optado por analizar las necesidades de cada elemento y mostrarlo de manera que estuviese representada toda su información.

4.5.2. Implementación de Funcionalidad y Dinamismo

Tras la esquematización y a partir de una versión estática de la aplicación se procedió a implementar la funcionalidad de los formularios y hacer que la información fuese dinámica y obtenida de la base de datos. Para ello se ha utilizado Angular y Spring. Figura 4.18.

Front-End

Llamamos *front-end* a la parte de la aplicación encargada de mostrar y representar la información. Para ello se usan clases servicio similares a las del *back-end*. En ella se ve representada la parte de la vista del MVC.
La aplicación está estructurada en componentes, servicios y estructuras de datos representativas de cada entidad.

Componentes: Representan cada una de las posibles vistas o estructuras visuales de la aplicación. Están compuestos de tres archivos:

- *Componente.html*: Contiene el código HTML asociado al componente.
- *Componente.css*: Contiene el código que aporta estilo fuera de Bootstrap al componente.
- *Componente.ts*: Contiene el código TypeScript que da funcionalidad al componente.

Los componentes de la aplicación son:

- Footer: Establece el pie de página de la aplicación.

- Form-Registro-Usuarios: Contiene el formulario de registro donde recoge todos los datos necesarios para la el registro de los usuarios. Es una clase simple, que contiene la variable de formulario y una función de registro a través de la que llama al servicio y guarda el usuario en la base de datos.

- Home: Muestra las tablas con las órdenes de pago del usuario logueado. Su funcionalidad principal es la carga de datos. En cargarOrdenesUsuario() carga todas las órdenes emitidas por el usuario y las separa en los cuatro estado posibles. En cargarOrdenesIP() recoge todas las órdenes en las que el usuario participa como investigador principal para poder tramitarlas. Además de estas funciones, contiene otras correspondientes a paginación y aspectos visuales de la aplicación.

- Login: Este componente se encarga de administrar la sesión de usuario y comprobar que existe antes de darle acceso a la aplicación. Su función principal es login(). Para mantener la sesión se usa una variable propia de los navegadores, *sessionStorage*, que se administra desde el servicio de sesión.

- Menu-Vertical: Carga el menú lateral de la web que que enlaza con las distintas vistas.

- Navbar: Establece la cabecera de la web y contiene la función que permite cerrar la sesión.

- Perfil: Es el componente encargado de la vista con su mismo nombre. Las funciones principales de este componente son las de modificación del perfil, que permiten cambiar los datos del usuario logueado. Estas funciones son: modificarIban(), que se encarga de crear un nuevo acreedor con los datos del usuario y el iban introducido a través de la vista o modificar el existente, modificarUsuario() que administra los datos del usuario y modificarDatos() que aúna las dos anteriores.

- Vista Orden: La vista de orden está formada por seis componentes.

    - Add-orden: Se encarga de recoger los datos necesarios para generar una orden de pago de gastos generales. Tiene dos funciones principales. La función encargada de crear cada uno de los gastos, crearGasto() y la función encargada de crear la orden de pago, crearOrden(). Además de estas funciones contiene otras encargadas de cargar algunos datos necesarios como los proyectos o acreedores que existen en la base de datos.
    - Add-orden-viajes: Es el equivalente a add-orden para gastos de viaje. Funciona igual pero recoge además los campos correspondientes de las órdenes de viaje.
    - Editar-orden: Sirve para editar una orden ya emitida, independientemente si es de viajes o general. Su función principal es editarOrden(). Esta función tambien se encarga de cambiar el estado de la orden para volver a enviársela a los investigadores principales correspondientes para su firma.
    - Ver-orden: Permite visualizar la orden de manera individual y administrar el estado según el usuario logueado. Sus funciones principales son: cargarOrden() que obtiene los datos de la propia orden y sus gastos, la función encargada de modificar el estado, cambiarEstadoOrden(*estado*) que modifica la orden que se está visualizando, y la función generarPDF() que se encarga de llamar al

servidor, recoger el archivo generado por este y tramitarlo para que pueda ser visualizado por el usuario.

- Vista-orden-boton: es un componente que hace de pasarela entre la tabla de visualización de órdenes y crear orden, donde se permite escoger que tipo de orden se va a crear. Si en un futuro se añadiesen más tipos de órdenes habría que redirigirlo desde aquí.
- Vista-ordenes: Se encarga de traer el listado de órdenes emitidas por el usuario. No posee funciones especiales. Las más importantes son la que llama al servidor para traerse las órdenes y las funciones de ordenación.

- Vista-acreedores: Esta vista contiene dos componentes:
  - Form-acreedores: Es el componente encargado de crear y editar acreedores. Sus funciones principales son crearAcreedor() y actualizarAcreedor().
  - Ver-acreedores: Solo tiene una función destacables que es la que se encarga de cargar todos los acreedores de la aplicación para luego mostrarlos.
- Vista-proyecto: Esta vista contiene tres componentes:
  - Form-proyecto: Desde este componente se genera el formulario necesario para la creación de un proyecto. Su función más destacable es crearProyecto() que tras las comprobaciones necesarias inserta el nuevo proyecto en la base de datos.
  - Ver-Proyecto: Es el componente encargado de mostrar los datos, integrantes y órdenes de pago de un proyecto. Además permite modificar algunos de estos datos al IP del proyecto o añadir nuevos participantes. Tiene varias funciones destacables como son: cargarProyecto(), cargarUsuariosProyecto() y cargarOrdenesProyecto() que se encargan de la carga desde la base de datos. La función modificarProyecto() permite cambiar los datos referentes a este. Las funciones anadirInvestigador() y eliminarInvestigador() añaden y eliminan investigadores al proyecto respectivamente.

Servicios y estructuras de datos: Los servicios son los encargados de la comunicación con el servidor. Las estructuras de datos definen la forma que tendrá cada una de las entidades que se van a transmitir entre el *front-end* y el *back-end* y a las que se va a dar uso en la aplicación.
Cada servicio tiene asociada una estructura excepto la sesión que da servicio a la propia aplicación. Los servicios de la aplicación son: acreedor, gasto, gasto-viaje, orden, proyecto, usuario, usuario-proyecto y sesión. Cada cual se comunica con su respectiva clase REST del *back-end* y la estructura posee los mismos campos que la entidad de este.

Back-End

El *back-end* es la parte de la aplicación encargada de la comunicación con el servidor. Sirve para obtener los datos de la base de datos y ponerlos a disposición de la aplicación a través de un archivo JSON.
Para tener acceso a los datos se requirió implementar el MVC. En el *back-end* quedan representados el modelo y el controlador.

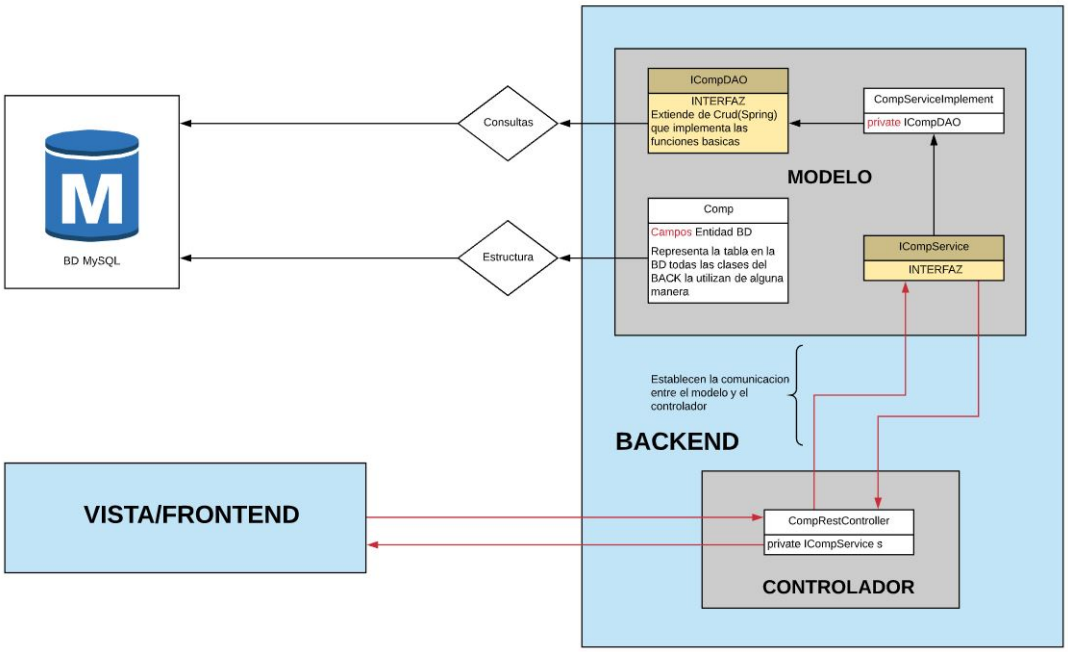


Figura 4.18: Ejemplo de la entidad proyecto

Creación de las entidades: Cada tabla de la base de datos requiere una clase que represente la entidad con cada uno de sus campos y el tipo de estos. Las entidades y sus campos son:

- Acreedor: iban, nif y nombre.
- Gasto:id_orden, nfactura, importe, descripcion, iva y foto.
- Orden: id, acronimo, numeracion, estado, nif_user, fechaOrden, num_contabilidad, concepto, memoria, relacion, nif_acreedor y observaciones.
- Proyecto: acronimo, nombre, presupuesto, nContabilidad, ip1, ip2, fechaInicio y fechaCierre. Se puede ver en la Figura 4.19.
- Usuario: dni, password, nombre, apellido1, apellido2, email, telefono, departamento y centro.
- UsuarioProyecto: id, dni, acronimo y rol.

```java
@Entity
@Table(name = "proyecto")
public class Proyecto implements Serializable {

    @Id
    private String acronimo;

    private String nombre;
    private double presupuesto;
    private int nContabilidad;

    @Column(name="ip1")
    private String ip1;

    @Column(name="ip2")
    private String ip2;

    @NotNull(message="No puede estar vacio")
    @Column(name="fecha_inicio")
    private Date fechaInicio;

    @NotNull(message="No puede estar vacio")
    @Column(name="fecha_cierre")
    private Date fechaCierre;

    public String getAcronimo() {
        return acronimo;
    }

```

Figura 4.19: Ejemplo de la entidad Proyecto

Creación de la clase DAO(*Data Access Object*) : La clase DAO es la encargada de la comunicación entre el *back-end* y la base de datos. Cada entidad tiene su clase DAO correspondiente la cual extiende de CrudRepository que contiene las consultas básicas. Además mediante una función con su respectiva anotación *@Query* se pueden generar consultas más específicas con parámetros recibidos desde la aplicación. Figura 4.21.

```java
@Query(value ="SELECT * FROM proyecto as p where p.acronimo in "
        + "(select u.acronimo from usuarioproyecto as u where u.dni= :d )", nativeQuery=true)
public List<Proyecto> getProyectosUsuario(@Param(value = "d")String dni);
```

Figura 4.20: Ejemplo de la entidad proyecto

Creación de la clase REST: La clase REST de cada entidad sirve para establecer la comunicación entre el *front-end* y el *back-end*. Recibe las peticiones del servicio del *front-end* y las tramita desde el lado del servidor para obtener una respuesta que será de vuelta. Figura 4.22.

```java
//Va a retornar los proyectos en los que participa un usuario (como IP o investigador)
@PostMapping(path="/proyectosUsuario")
public List<Proyecto> getProyectosUsuario(@RequestBody String dni) {
    return proyectoService.getProyectosUsuario(dni);
}
```

Figura 4.21: Ejemplo de la entidad proyecto

Creación de la interfaz de servicio e implementación: Cada entidad requiere su propio servicio. Para ello se ha creado una interfaz que establece las funciones a implementar y una clase que implementa esta interfaz, que sera llamada desde la clase REST de cada entidad e invocará a la clase DAO para establecer la comunicación con la base de datos. Figura 4.20.

```java
@Service
public class ProyectoServiceImplement implements IProyectoService {
    @Autowired
    private IProyectoDao proyectoDao;

    @Override
    public Proyecto insertarProyecto(Proyecto proyecto) {
        return proyectoDao.save(proyecto);
    }
```

Figura 4.22: Ejemplo de la entidad proyecto

4.6. Desglose de Funciones de Especial Interés

4.6.1. Creación de PDF

Para crear los archivos PDF correspondientes a cada orden de pago, hemos utilizado la biblioteca PDFBox. Las funciones que hacen uso de ella están implementadas en la clase OrdenRestController.java del *back-end*. Aunque hay seis funciones dedicadas a ello debido a los dos tipos de formulario a rellenar, se pueden resumir en cuatro:

- rellenarDatosIP: Dado un usuario rellena los campos del IP que firma la orden. Al ser la primera llamada también se encarga de abrir el archivo de la url correspondiente y generar uno auxiliar que se va rellenando mediante la ejecución de distintas funciones.
- rellenarGastos: Recibe los gastos asociados a la orden rellena y los campos correspondientes a estos.
- generarPDF: Es una función común a todas las órdenes, que se encarga de rellenar los datos comunes de estas y generar el archivo final.
- mostrarPDF: Toma el nombre del archivo que tiene que mostrar y lo transforma en un tipo de dato procesable por TypeScript que se encarga de mostrarlo al usuario de la aplicación.

  Las funciones de tratamiento de los campos son 'getField(*nombreDelCampo*)' para acceder al campo con el nombre indicado y 'setValue(*nuevoValor*) para establecer el nuevo valor de este. También se utilizan 'PDDocument.load(*archivo*)' para cargar el PDF, 'pdfDocument.getDocumentCatalog().getAcroForm()' donde pdfDocument es el archivo cargado, para generar una estructura de datos que contiene los campos asociados a una clave y las funciones 'save()' y 'close()' para guardar y cerrar el archivo respectivamente. Figura 4.23.

```java
@RequestMapping(value = "/mostrarPDF/{id}", method=RequestMethod.GET,  produces = MediaType.APPLICATION_PDF_VALUE)
public ResponseEntity<byte[]> mostrarPDF(@PathVariable("id")Long idOrden){

    Path rutaArchivo = Paths.get("pdfs").resolve("auxPdf.pdf").toAbsolutePath();

    File f = new File(rutaArchivo.toString());
    f.getAbsolutePath();
    InputStream inputStream;
    try {
        inputStream = new FileInputStream(f.getAbsolutePath());
         //String inputStreamToString = inputStream.toString();

        byte[] content =   IOUtils.toByteArray(inputStream);
        return new ResponseEntity<>(content, this.getPDFHeaders("auxPdf.pdf"), HttpStatus.OK);

    }catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}

public HttpHeaders getPDFHeaders(String fileName) {
    HttpHeaders head = new HttpHeaders();
    head.setContentType(MediaType.parseMediaType("application/pdf"));
    head.add("content-disposition", "attachment; filename=" + fileName);
    head.setContentDispositionFormData(fileName, fileName);
    head.setCacheControl("must-revalidate, post-check=0, pre-check=0");
    return head;
}
```

Figura 4.23: Función Mostrar PDF

4.6.2. Validación de Formularios

En el código del *front-end* de la aplicación nos aseguramos de validar los formularios antes de enviarlos al servidor. Para ello usamos las clases FormBuilder, FormGroup y Validators de Angular, que permiten establecer un nombre para cada uno de los *inputs* del código html del componente para indicarle a que campo se refiere y que requisitos tiene que cumplir. En la Figura 4.24 se puede ver el código donde se establecen los validadores. La mayoría de los validadores vienen preestablecidos, como por ejemplo un formato de e-mail correcto o un tamaño máximo del campo. También se permite un tipo *pattern* en el que podemos establecer ciertos patrones que debe cumplir la cadena de texto introducida. En general este tipo de patrón se puede ver en los constructores de los componentes de creación de elementos como son form-registro, add-orden, form-proyecto etc. El esquema que sigue cada uno de los campos es:

nombre: [‘Valor Inicial’, [Validación.tipo1, Validación.tipo2 ...]]

```
this.formRegistro = this.fb.group({
  dni: ['', [Validators.required, Validators.pattern, Validators.minLength(9)]],
  nombre: ['', [Validators.required]],
  apellido1: ['', [Validators.required]],
  apellido2: ['', [Validators.required]],
  email: ['', [Validators.required, Validators.email]],
  password: ['', [Validators.required, Validators.minLength(5)]],
  telefono:['', [Validators.required]],
  departamento: [''],
  centro:[''],
  tyc:['', Validators.requiredTrue]
});
```

Figura 4.24: Creación del Formulario.

4.6.3. Paginación de Tablas

Para la paginación de tabla, se han diseñado un conjunto de funciones que se encargan de generar pequeños arrays con los índices de los elementos del array principal. Esto permite escoger en cada vista cuantos elementos por página hay que mostrar y asi facilitar la creación de filtros con un función que permite escoger este número. Para estas funciones se requieren tres variables: una para establecer el numero de elementos por página, otra para almacenar la pagina en la que estamos en ese momento y una matriz de elementos de la forma elementos[*tamañoPaginaActual*][*tablaAMostrar*].
Las funciones necesarias para el paginado son:

- inicializarArrayNElementos(*nElementosPorPágina*, *tablaAPaginar*): donde crea el array del tamaño adecuado para la página que tiene que mostrar.
- anterior/siguiente(*páginaActual*, *longitudDelArrayPrincipal*, *tablaAPaginar*): Devuelve la pagina a mostrar con su array de elementos correspondiente.
- getOrdenPaginadoIndex(*posición*, *páginaActual*): Devuelve el índice del array principal correspondiente a la posición del paginado recibida.

4.6.4. Ordenación

Cada una da las tablas puede requerir ser ordenada. Para ello se han creado funciones de tipo *sort* que se implementan para cada elemento por el que queremos ordenar. Figura 4.25.

```
orderByFechaI():void{
  this.proyectos.sort(
    function (a, b) {
      if (a.fechaInicio> b.fechaInicio) return 1;

      if (a.fechaInicio < b.fechaInicio) return -1;

      return 0;
    });
}
```

Figura 4.25: Función de ordenación de los proyectos por fecha inicial

4.7. Desglose de Ficheros de Especial Interés

4.7.1. pom.xml

El archivo pom.xml es un fichero principal de un proyecto Maven, perteneciente a la parte *back-end*, que contiene información acerca del proyecto. Detallaremos las dependencias instaladas:

- Web: Es una de las más importantes ya que nos permite crear nuestras clases controlador utilizando la etiqueta @RestController. El controlador se comunica con el servicio que a su vez se comunica con el modelo propio de cada entidad, los datos en el controlador se envían y se reciben en formato JSON.
- SQL - JPA: Para utilizar nuestras clases entidad que serán las tablas de la base de datos.
- MySQL: Para trabajar con la base de datos.
- Core - DevTools: Permite que los cambios que se realizan durante la programación, se actualicen en el servidor automáticamente sin tener que lanzar de nuevo el servidor, es decir, hace el despliegue automáticamente.
- PDFBox: Para crear los archivos PDF correspondientes a cada orden de pago. Las funciones que hacen uso de esta están implementadas en la clase OrdenRestController.java.

4.7.2. application.properties

Es un archivo perteneciente a la parte *back-end*. A continuación se detallan las configuraciones necesarias para lanzar la aplicación en el servidor local la realización del despliegue, así como los requisitos del tamaño de las fotos.

4.7.3. config.ts

Lo encontramos en el *front-end* dentro de la carpeta config. Este archivo tiene dos rutas, una para trabajar en local y otra para cuando se hace el despliegue al servidor.

4.7.4. app.module.ts

Este archivo lo encontramos en el *front-end* dentro de la carpeta app. En el se detallan todos los componentes y servicios que forman parte del proyecto, así como las rutas y los correspondientes componentes a los que hacen referencia. En imports se importan los módulos que utilizamos:

- BrowserModule: Utilizamos sus directivas en las vistas, por ejemplo la directiva ngIf sirve para elegir qué mostrar según la condición que tenga.
- FormsModule: Para la realización de formularios.
- ReactiveFormsModule: Nos permite introducir controles dentro de los formularios y asociarlos a etiquetas HTML sin usar ngModel.
- RouterModule: Nos permite fácilmente la creación de rutas hacia los componentes que forman la aplicación.
- HttpClientModule: Lo utilizamos para hacer llamadas a la API REST y obtener los resultados por defecto en JSON.

4.8. Despliegue de la aplicación web

Las aplicaciones webs se ejecutan en un servidor. En nuestro proyecto se utiliza para el *back-end* la plataforma que nos ofrece Heroku, y el *front-end*, Firebase.

4.8.1. Heroku

Heroku[5] es un Paas[5] que nos permite manejar el servidor y sus configuraciones. Su arquitectura es mucho más robusta y segura que un hosting.

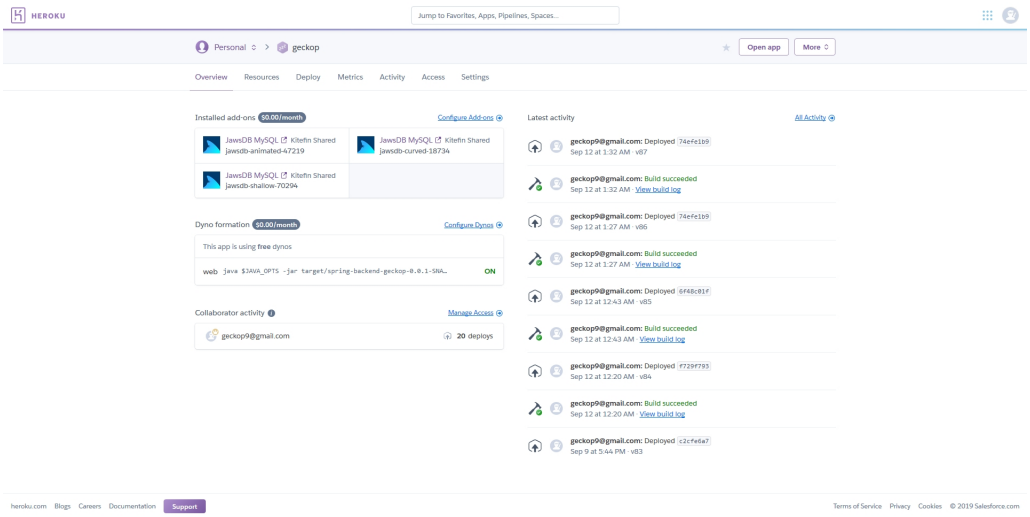
El proceso para el despliegue ha consistido en registrarnos en Heroku, crear un nuevo proyecto concretando como lenguaje java y creando una base de datos. A continuación, desde la consola, nos conectamos a Heroku haciendo login, creamos un proyecto, devolverá un host, usuario y contraseña de la base de datos.

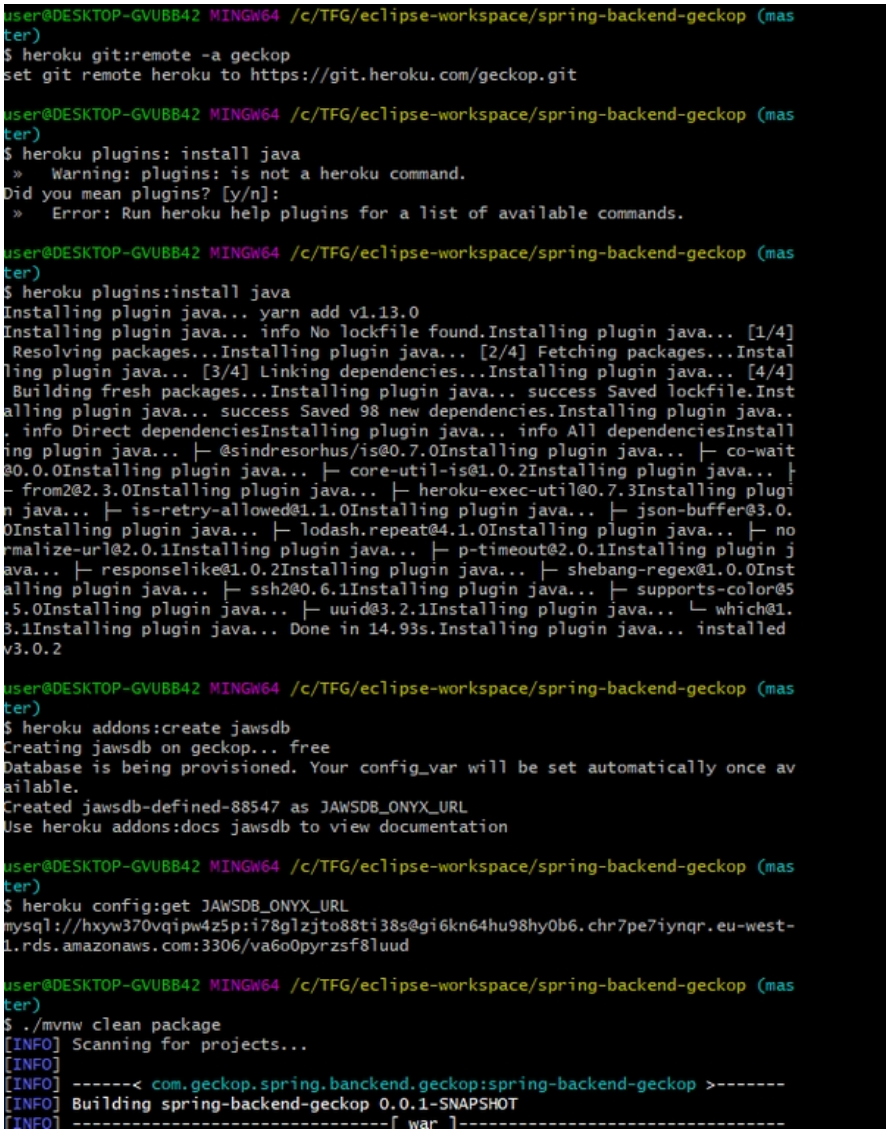
Estos datos tenemos que introducirlos en el archivo application.properties del *back-end* para realizar la conexión al servidor, en vez de manera local como hemos trabajado hasta el momento. Empaquetamos el contenido *back-end*, utilizando el comando ./mvnw clean package, este empaquetado tiene extensión war, es un archivo comprimido que contiene el código que compone el *back-end*.

Una vez creado el archivo war, realizamos el despliegue en Heroku. Si se ha realizado correctamente el deploy, nos devuelve la ruta para acceder al back del proyecto. Para comprobar que se ha subido correctamente, utilizamos la herramienta Postman[6], que nos permite realizar peticiones sobre la API de una manera muy sencilla asi como probar el correcto funcionamiento de la aplicación.

[5]Plataforma como servicio
[6]https://www.getpostman.com/

En este proyecto hemos realizado 20 despliegues del *back-end*. Figura 4.26.





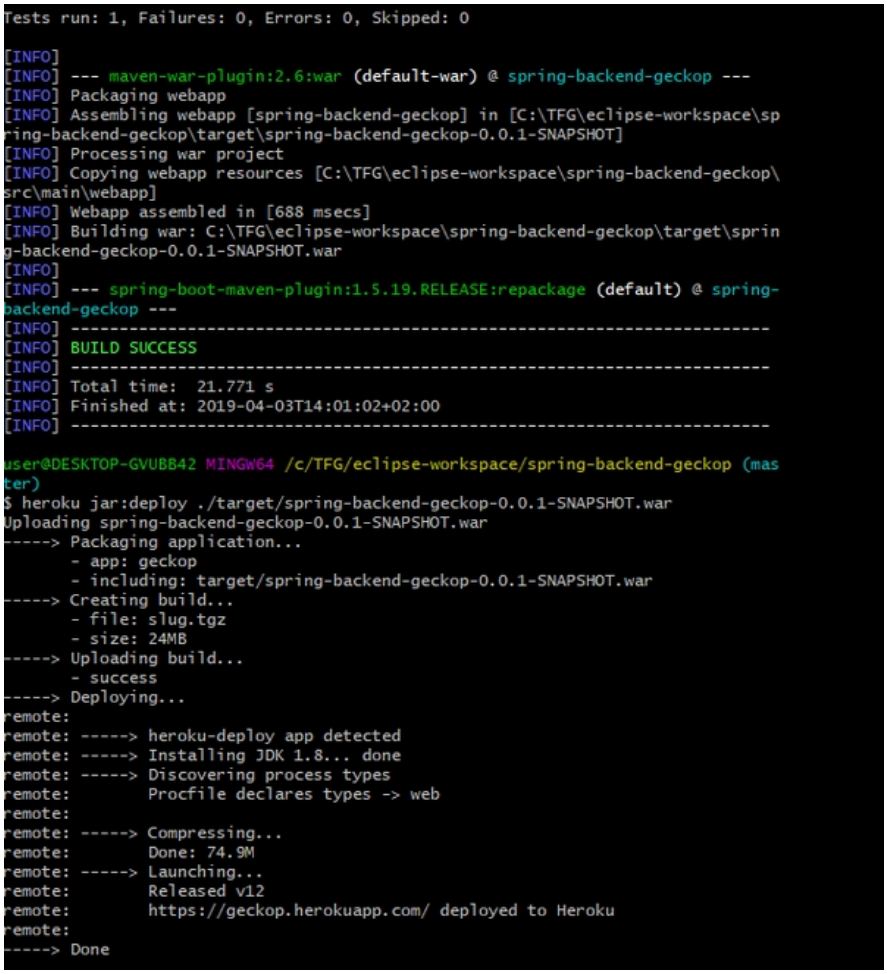


Figura 4.26: Despliegue realizado correctamente

4.8.2. Firebase

Firebase[4] es una API que nos permite guardar y sincronizar datos en la nube en tiempo real. El proceso de despliegue es muy parecido al del *back-end*. En este caso lo hacemos con el *front-end*. Tenemos que registrarnos en Firebase con una cuenta de Google y crear un nuevo proyecto. Luego desde consola nos logueamos a Firebase y procedemos a realizar los pasos para el deploy. En primer lugar, tenemos que empaquetar nuestro *front-end*, para ello utilizamos el comando ng build –prod. Después Firebase nos hará una serie de preguntas para hacer el deploy; elegiremos Hosting, el proyecto que hemos creado en Firebase y realizar el deploy. Si todo ha ido correctamente nos devuelve la ruta donde podemos ver la web. Figura 4.27.

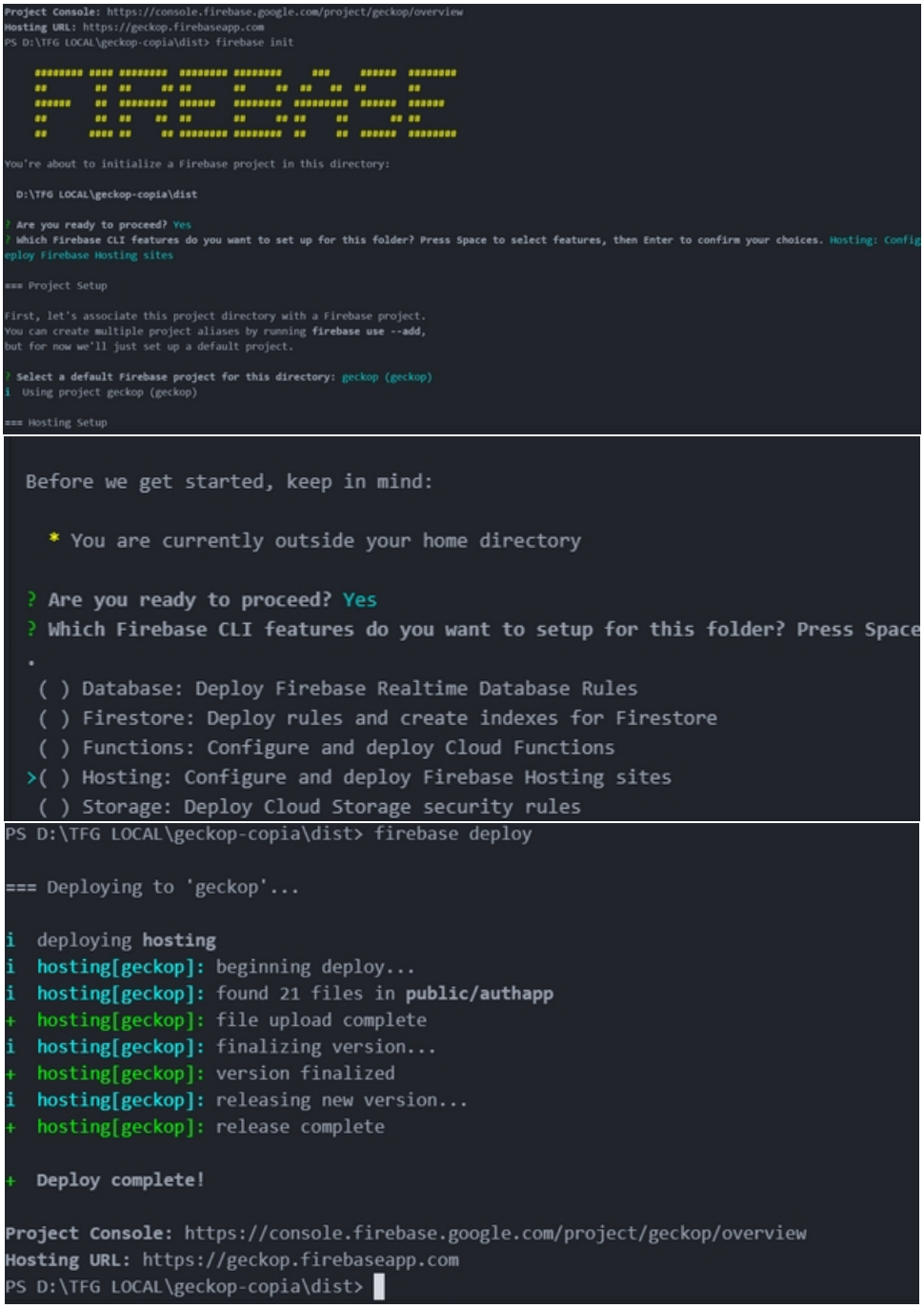


Figura 4.27: Despliegue realizado correctamente

4.9. Problemas Encontrados

Aunque se han encontrado muchos más problemas durante el desarrollo estos son los más destacables.

4.9.1. Configuración del Entorno de Desarrollo

Para la configuración del entorno de desarrollo, hubo que aprender como configurar cada una de las herramientas que un principio eran desconocidas. Desde la creación del

proyecto con sus dependencias correspondientes, la configuración del servidor local y la comunicación entre las distintas tecnologías, se necesitaron una gran cantidad de horas buscando información sobre todas las configuraciones necesarias para hacer que funcionase correctamente.

4.9.2. Directivas de Spring

Para la comunicación entre la vista y el servidor se requieren directivas características de Spring que al menor cambio producen fallos muy complicados de detectar y que al final del proyecto se corregían a base de práctica porque no se ha descubierto una manera eficiente de depurar los fallos.

4.9.3. Guardado de sesión

Puesto que el manejo de sesiones aprendido durante la carrera se realizaba mediante una variable propia de una tecnología especifica que no se ha usado en el desarrollo de la aplicación, hubo que buscar una alternativa. La encontrada fue la variable global de los navegadores SessionStorage que se encarga de ello.

4.9.4. Paginación

Con el uso de la aplicación en un estado de desarrollo avanzado se manifestó la necesidad de paginar la visualización de algunas tablas. Puesto que las soluciones ya desarrolladas por terceros requerían cambios muy grandes en el código ya implementado, se ha desarrollado un sistema de paginación de tablas propio con un uso más simple.

4.9.5. Generación de PDF

Al tratar de generar el archivo requerido por la Fundación UCM para solicitar el pago en las órdenes, nos encontramos con el problema de rellenar formularios pdf. Actualmente hay multitud de herramientas que permiten la generación de archivos pdf mediante Java pero la mayoría no permiten la edición de estos y las pocas que lo permiten tienen licencia privada y requieren de pago para su uso. Solo se ha encontrado una biblioteca que no requiera de este procedimiento y ha sido la utilizada en la implementación. Esta biblioteca sigue actualmente en desarrollo aunque la versión utilizada es estable y cumple con las funcionalidades necesarias.

Capítulo 5

Contribuciones al Proyecto

5.1. Azahara Fernández Notario

Durante la fase de análisis mi labor consistió en realizar reuniones con nuestra tutora para plantear el trabajo a realizar y como repartir las tareas que haríamos cada uno de los componentes. Además de hacer un diseño previo de como sería la base de datos que almacenase la información de la aplicación. Por otro lado realicé algunos de los diseños previos de las vistas (no funcionales) para poder plantear la cantidad de trabajo que tenía cada una.

Para mí la fase de investigación, consistió en formarme y hacer pruebas sobre las tecnologías que decidimos usar además de configurar mi entorno local del cual se encargó mi compañera de buscar la forma correcta de hacerlo y compartir la información. Proceso que fuimos documentando para mas tarde incluirlo en esta memoria. Además de esto me documente sobre tecnologías para mejorar el aspecto visual de la aplicación sin tener un diseñador detrás del trabajo y cuando escogimos Bootstrap como herramienta aprender su uso para la posterior fase de diseño.

La fase de configuración e instalación para mi fue la más laxa y en la que menos trabajo dediqué. Por consecuencia mi labor se basó en seguir con la fase de investigación y empezar con algunos de los diseños para la siguiente fase.

En el diseño de la aplicación me dediqué a realizar distintas versiones usando HTML y Bootstrap de lo que sería visualmente la aplicación y como enlazarían unas vistas con otras. Además diseñé un icono y busque un nombre que gustase al equipo. También escogimos un estilismo que usar de manera conjunta para evitar discrepancias que resaltasen en las distintas vistas y escoger el uso de colores que ayudasen a identificar los componentes de la página de forma clara y evitar un aspecto muy recargado.

La fase de desarrollo ha sido la más larga y costosa. Durante esta fase he ido desarrollando distintas partes de la aplicación. La vista de registro fue mi punto de partida, puesto que considere que es la primera vista que necesitaba la aplicación. Para poder empezar a trabajar se necesitaba poder almacenar los datos de los usuarios que iban a interactuar con la herramienta. Esta vista tuvo la peculiaridad de ser el primer formulario con varias comprobaciones al que me enfrentaba lo que me llevo a la investigación de la biblioteca

Forms, que contiene a parte de clases muy útiles para la recogida de datos del formulario la clase Validators que nos permitía cerciorarnos de que los datos introducidos cumplen con los requisitos.

Lo siguiente a desarrollar fue la vista de login. Al implementar esta vistá me encontré con el problema de almacenar los datos de sesión. Tras investigar sobre cual era el método más común para esta labor, descubrí la variable *sessionStorage* que posen los principales navegadores (Google Chrome, Firefox, Safari, Edge, etc) obteniendo así una solución para el problema.

Tras este trabajo pase a desarrollar la pantalla de perfil. No fue una tarea especialmente complicada puesto que venia de tratar la recogida de datos del usuario. El único problema encontrado fue la recogida del IBAN para establecer al usuario como un acreedor mas de la aplicación que decidimos posponer hasta que el módulo de acreedores funcionase correctamente y estuviese probado.

El trabajo más largo a desarrollar fue la continuación del módulo de proyecto, que aunque mi compañera había iniciado la recogida de los datos principales, tras una reunión con la tutora nos dimos cuenta de que habíamos dejado una parte importante sin tener en cuenta y acabé terminándola yo. Tras la recogida de datos procedí a la elaboración de la visa en singular de un proyecto. Esta vista fue compleja en cuanto a tener que obtener información de todos los módulos de la aplicación, desde el usuario que esta logueado para ver si tiene o no permisos de acceso y edición hasta las ordenes de pago asociadas a él.

Tras tener terminado el módulo de proyectos y mi compañera tener en un estado muy avanzado el de ordenes pude empezar con la pantalla principal. En esta pantalla me encontré con el problema de tener que mostrar mucha información en poco espacio, que con ayuda de la tutora resolvimos.

Para terminar con el desarrollo me dediqué a la firma de ordenes, que fue el proceso mas complejo no por el cambio de estado de la orden si no por la generación del archivo pdf asociado a esta. Encontrar la biblioteca que nos permitió realizar esta tarea fue sumamente complicado, incluso planteamos excluirlo del alcance del trabajo por falta de una herramienta para hacerlo, aunque una vez encontrada la biblioteca la implementación fue sencilla.
Además de esto realice las funciones de paginación y ordenación de todas las tablas de la aplicación.

Con esto dimos por finalizada la fase de desarrollo. Se puede ver en las estadísticas de GitHub los casi 150 commits realizados entre la parte de *front-end* y *back-end* y aproximadamente diez mil lineas de código escritas, como se puede ver en la Figura 5.1 a la izquierda quedan las estadísticas del *front-end* y a la derecha las del *back-end*

Figura 5.1: Estadísticas Github.

Para finalizar el trabajo y con toda la documentación recabada a lo largo de la realización del trabajo procedí con la redacción de esta memoria junto con mi compañera.

5.2. Isabel Núñez de la Torre

Al inicio del curso 2018/2019 y comenzando el TFG, analizamos junto con la tutora la aplicación que tenían realizada anteriormente para la gestión de los proyectos de investigación, vimos que se necesitaba modernizar y ampliar funcionalidad.

Primero sacamos una versión previa de la base de datos con las acciones que se realizaban en la aplicación, así como extras que se querían realizar y no formaban parte de la ya existente. Realizamos mockups de las vistas previamente, para utilizarlos como base en el diseño de la aplicación.

Tras esta primera fase de análisis, investigué las mejores herramientas con las que podíamos llevar a cabo el TFG, puesto que la tutora nos dio total libertad al elegir las tecnologías con las que desarrollarlo, en un primer momento pensamos desarrollar la aplicación con las tecnologías vistas en la carrera: HTML, CSS, JS, PHP. Pero al tratarse de un trabajo de fin de grado y tener una parte de investigación, buscamos tecnologías que se estaban utilizando en el desarrollo de aplicaciones web en las empresas, vimos que las más demandadas eran Angular y Spring principalmente junto con las nociones básicas de las otras tecnologías vistas en la carrera. Contemplando las ventajas que podían ofrecer a nuestro TFG respecto a robustez y seguridad, así como a nivel de separación de la parte visual y lógica, incluso poder modularizarlo para un fácil mantenimiento posterior, elegimos Angular y Spring para el desarrollo junto con MySQL para la parte de creación y gestión de la base de datos.

Busqué todas las configuraciones e instalaciones de los entornos necesarios para empezar a programar, como entorno para la parte *front-end* utilicé Atom con Angular y para la parte *back-end* el entorno de desarrollo de Eclipse con Spring y Maven, para trabajar con la base de datos descargué MySQL Worbench.

Una vez configurado e instalado todo correctamente, procedí a la creación de los proyectos, y un repositorio en Github con su correspondiente subida.

Durante estos primeros meses nos encontramos con una dificultad, uno de los integrantes del grupo se desconectó del proyecto, lo que supuso un retraso y una sobrecarga de trabajo, se llevó a cabo la reorganización de las tareas.

Comencé a desarrollar el módulo de Acreedores tanto en el *back-end* como en el *front-end*, puesto que era el módulo de menor dimensión y para coger soltura con el manejo de las distintas herramientas, así como comprender la conexión entre la parte visual y lógica me pareció lo más práctico empezar por él.

Una vez analizada la estructura para programar en el *back-end*, entendiendo la separación en: entidades, modelo, servicio, controlador y como se comunicaban entre sí, seguí con la vista de proyectos, qué después continuaría desarrollando más en profundidad mi compañera con los cambios que surgirían tras las reuniones con la tutora.

Programé las rutas, configurando el archivo app.modules.ts y utilizando las librerías Router y ActivatedRoute, una vez que tuvimos forma de navegar por las distintas páginas

de la web, seguimos con el desarrollo de las distintas partes, me centré en las Órdenes, se nos pidió dos órdenes de pago distintas obtenidas de la Fundación de la UCM, comencé con el desarrollo de la orden de gastos generales, primero con la base de datos, es decir la parte *back-end*, una vez realizada esta parte y probando que las consultas funcionaban correctamente, pasé al desarrollo del *front-end* de la misma orden.

Me gustaría destacar que TypeScript, lenguaje con el que hemos programado la parte visual de la aplicación, es asíncrono por lo que aquí la fase de depuración del código fue clave, ya que al estar acostumbrada a trabajar con lenguajes síncronos, no fue simple y dio muchos quebraderos de cabeza porque no se recogían los valores en muchas ocasiones y quedaban como valores no definidos, esto ocurría al guardar en variables datos que se necesitaban entre distintos componentes como por ejemplo gastos y orden, puesto que había que saber identificar qué gastos pertenecían a una orden. Cuando había que editarla, como es asíncrono, al querer guardar el identificador de la orden en los gastos, los gastos tenían que estar dentro de donde se recogía la información de la orden ya que, si estaba en otra función, ese valor no estaba definido.

Después realicé el tratamiento de imágenes. Introducirlas llevó mucha parte de investigación y depuración para su correcto funcionamiento, incluir que no se pudieran añadir otros archivos que fueran otro formato distinto al de imágenes por la seguridad de la aplicación y restringir el tamaño máximo que podría tener una imagen para no saturar el espacio en la base de datos. El siguiente paso fue generar una tabla a partir de una consulta para que se mostraran únicamente las órdenes del usuario que estaba registrado. Dado que la aplicación está modularizada, realizar una consulta conlleva añadir código en seis clases, dos en la parte *front-end* y cuatro en la parte *back-end*.

Una vez terminada esta orden, pasé a desarrollar la vista de órdenes de gastos de viajes, tenía similitudes con la anterior, exceptuando los gastos que eran distintos, esta orden al ser más compleja por la gran cantidad de datos que tiene, estuvimos mi compañera y yo con ella sobre todo en la parte de depuración de la misma. Una vez completadas ambas órdenes, me puse con el editar las órdenes.

A lo largo del TFG hice varios despliegues a dos servidores, los servidores elegidos fueron Firebase para la parte *front-end* y Heroku para la parte del *back-end*, ambos los elegí tras investigar sobre servidores por: robustez, seguridad y si tenían una versión gratuita.

Investigué y aprendí a configurar desde consola como realizar cada despliegue para que funcionara bien, había que tener mucho cuidado en realizar todos los pasos detalladamente porque si un paso se omitía, aunque ponía despliegue realizado con éxito luego al verlo desde la web se podía comprobar que no se había subido correctamente.

Busqué información sobre los términos de uso y políticas de privacidad para enlazar en el registro, decidí junto a mi compañera que lo mejor era enlazar a las políticas de la propia UCM dado que el proyecto se subiría a un servidor suyo en un futuro y tendría que cumplir sus políticas.

Destacar que ha sido todo un reto utilizar tecnologías hasta entonces nunca vistas ni utilizadas, mucho tiempo invertido en depuración de fallos y problemas en la realización de ciertas tareas como rellenar automáticamente el PDF desde la base de datos, el manejo de imágenes, etc. Puesto que, al ser tecnologías relativamente recientes, falta mucha información del uso de las mismas por Internet.

Aproximadamente las líneas programadas han sido 9000 - 10000. Figuras 5.1 y 5.2

Figura 5.2: Estadísticas Github *front-end*.

Figura 5.3: Estadísticas Github *back-end*.

Capítulo 6

Trabajo Futuro

La aplicación aunque está terminada de manera funcional y cumple los requisitos establecidos presenta varias posibilidades de ampliación y mejora.

6.1. Módulo de Contabilidad de los Proyectos

Sería interesante para la aplicación añadir un módulo que permitiese revisar la contabilidad de los proyectos, calculando el presupuesto gastado, haciendo estadísticas anuales o diferenciadas por el tipo de orden y gastos.

6.2. Modulo de Congresos

Puesto que también se dan situaciones en las que los investigadores pueden organizar congresos, sería interesante añadir nuevas entidades que permitiesen gestionar los aspectos económicos de este tipo de eventos a través de la aplicación.

6.3. Alertas Via Email

Como mejora ante las notificaciones ya implementadas en la aplicación. Sería interesante añadir una función que alertase a los investigadores de los cambios de estado que se dan en sus órdenes a través de un correo electrónico.

6.4. Migración a Dispositivos Móviles

Junto con las alertas por correo sería interesante hacer una migración de la aplicación a una plataforma que permita consultar los movimientos desde dispositivos móviles. Haciendo la aplicación más portátil.

Capítulo 7

Conclusiones

Tras el trabajo de investigación, análisis, diseño, y desarrollo de la aplicación se puede concluir que el propósito de mejora estética esta especialmente logrado, ya que el nuevo diseño es mucho más intuitivo y limpio, haciendo así más simple localizar los datos de interés.
Se ha logrado la funcionalidad de generación de las órdenes de pago en formato PDF de manera automática y sencilla mediante un formulario web, lo que era un requisito necesario, y ha requerido un trabajo específico de investigación y aprendizaje de uso de una biblioteca concreta para ello.
Además se han renovado tecnológicamente todas las funcionalidades de la aplicación logrando una mayor facilidad a la hora de realizar el mantenimiento de la aplicación utilizando frameworks punteros de fabricantes de prestigio, lo cual da mayor seguridad en cuanto a su buen funcionamiento y adaptación a distintos entornos futuros o ampliación de las funcionalidades de la aplicación.

Capítulo 8

Conclusions

After the investigation, analysis desing and development of the aplication we could conclude that the purpose of the visual improvement is especially accomplised, as the new design is clearly more intuitive and clean, making it easier to spot the essential information. It has been achieved the functionality in the creation of the payment orders in PDF format as an automated and easy way throug a simple web application form, since it´s a requirement needed and which has required an specific work of research and use of a specific library for that purpose. Furthermore, it has been updated technologically most of the tools of the aplication using the latest frameworks from renowned manufacturers, which implies a greater reliability of its performance and adaptability in diferent environments later on or the extension of its functionalities.

Índice de figuras

Bibliografía

[1] Roger Dudler. https://rogerdudler.github.io/git-guide/index.es.html. Desconocido.

[2] The Apache Software Foundation. https://pdfbox.apache.org/. 2009.

[3] Google. https://angular.io/api/core. 2010.

[4] Google. https://es.wikipedia.org/wiki/Firebase. 2011.

[5] Salesforce.com. https://devcenter.heroku.com/articles/git. 2007.

[6] Pivotal Software. https://spring.io/projects/spring-boot. 2019.

[7] Bootstrap Team. https://getbootstrap.com/docs/4.3/getting-started/introduction/. 2011.

[8] SweetAlert2 Team. https://sweetalert2.github.io/. Desconocido.

PASCAL
ENERO 2018
Ult. actualización 19 de septiembre de 2019
LaTeX lic. LPPL & powered by TEFLON CC-ZERO